Easy Delphi

Introdução à API do Windows

Introdução ao desenvolvimento multicamadas

Clube Delphi +PHP

Ano 8 · Edição 97 · R$9,90

DevMedia

RTTI

Conheça na prática um dos recursos mais poderosos da linguagem Delphi

ISSN 1517990-7

NA WEB

Confira no portal ClubeDelphi PLUS 4 vídeo-aulas sobre Delphi for PHP e impressão matricial

Mini-Curso
Datacar OO - Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 2

Mini-Curso
Veja como criar um sistema on-line de controle para uma vídeo-locadora, com ASP.NET – Parte 5

Delphi for PHP
15 dicas para você resolver problemas comuns e turbinar seu Web Site PHP

Novidades
Explore o GridView, o controle mais robusto do ASP.NET 2.0 do RAD Studio 2007

Ask the Expert
DataSnap no Delphi 2006

PHP
Entendendo os métodos GET e POST

developers webdays 2008

100% de aproveitamento!

22 e 23 de agosto – SP

- Conheça as principais novidades das tecnologias **Java e .NET**
- Palestras sobre **Boas práticas e Arquitetura**
- Certificado de Participação
- **100% de aproveitamento** - Apesar das palestras acontecerem simultaneamente, não haverá perda de conteúdo. No dia do evento você receberá um DVD com o conteúdo completo de todas as palestras em formato de vídeo-aula!

São + de 46 horas de conteúdo.

Principais Temas:

Java	.NET	Extras
Ajax	Linq	SOA
Java FX	WPF	WEB 2.0
Grails	Silverlight 2.0	WEB 3.0
Android	ASP.NET 3.5	Arqutetura Web
Groovy	Ajax	Scrum
Jboss Seam	ASP.NET MVC	Microsoft XNA
JSF	Visual Studio 2008	Ruby on Rails
RIA (Flex, etc)	Novidades C# 3.0 e VB.NET 9.0	Domain Driven Design
Best practices em java	Best practices em .NET	Mashups

Maiores informações:
www.devmedia.com.br/webdays2008
evento@devmedia.com.br
(21) 3382-5025

Inscreva-se já!

Realização DevMedia group

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Delphi

QuickUpdate

08 – ASP.NET 2.0
Uma revolução em novidades para o desenvolvimento Web
[*Guinther Pauli*]

Expert

10 – RTTI
Conheça na prática um dos recursos mais poderosos da linguagem Delphi
[*Gustavo Chaurais*]

Web
Novidades

20 – GridView no RAD 2007
Dicas de como usar o Grid turbinado do ASP.NET 2.0
[*Luciano Pimenta*]

Mão na Massa
Mini-Curso e Web

28 – Controle on-line de vídeo-locadora - Parte 5
Veja como criar um sistema on-line de controle para uma vídeo-locadora
[*Maikel Marcelo Scheid*]

Boas Práticas
Mão na Massa
Mini-Curso e Expert

34 – Sistema DataCar em OO
Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 2
[*Paulo Quicoli*]

Easy Delphi

Easy Delphi

44 – Introdução à API
Entenda e saiba como utilizar Interface de Programação de Aplicativos (API) em suas aplicações
[*Adriano Santos*]

Easy Delphi

50 – Introdução à multicamadas
Aprenda os conceitos e como construir suas primeiras aplicações distribuídas
[*Guinther Pauli*]

PHP

Delphi for PHP

56 – Dicas de PHP
Aprenda dicas e macetes para usufruir ao máximo os recursos do Delphi for PHP
[*Fabrício Desbessel*]

PHP

62 – Entendendo os métodos GET e POST no PHP
Entenda e saiba como enviar e receber informações entre páginas usando os métodos GET e POST
[*Fabrício Desbessel*]

Você percebeu os nomes ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas conseqüência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em uma aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "Recordset", mostre como usar o Recordset de forma inteligente. para criar um carrinho de compras.

[Mini-Curso] Artigo que faz parte de um mini-curso.

[Delphi for PHP] Artigo com foco na versão PHP do Delphi.

[Boas Práticas] A Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas, mas também as melhores práticas – Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado.

[Easy Delphi] Com foco no desenvolvedor iniciante.

[PHP] Artigo com foco na linguagem PHP (não necessariamente utilizando o Delphi PHP).

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte do dia a dia do desenvolvedor mas que prometem ser a próxima febre.

[Boa Idéia] A Esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma 'boa idéia', dentro do contexto de desenvolvimento de software.

Clube Delphi + PHP

Ano 8 - 97ª Edição - 2008 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Adolfo Sabino
simininu@yahoo.com.br

Capa
Vinicius O. Andrade
viniciusoandrade@gmail.com

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

EDITORIAL

Certa vez criei uma unit daquelas tipo "faz tudo" (quem nunca criou sua util.pas que atire a primeira pedra). Lembro-me que existia um procedimento genérico que recebia um objeto como parâmetro e modificava uma determinada propriedade de nome fixo, Foo por exemplo. Esses objetos podiam ser de diversos tipos. Como bom conhecedor do Object Pascal, pensei de cara: declaro como parâmetro um tipo que seja comum (ancestral) a todas as classes que irei passar. Não adiantou, porque os objetos eram de tipos que possuíam diferentes ancestrais. Eu poderia usar RTTI e passar um TObject, testar um por um todos tipos, com o is e as e fazer um typecast, mas isso causaria uma forte dependência além de várias dezenas de ifs. Pensei outra alternativa, faço todas as minhas classes implementarem uma interface IFoo e declaro meu parâmetro do método da unit para aceitar um IFoo. Não daria, existiam classes que eu não podia modificar e aquilo seria mais POG (Programação Orientada a Gambiarra) do que POO. Foi aí que me lembrei da unit Typinfo. Yes! Então, fui de cara verificar a procedure SetPropValue, que recebe como parâmetro um objeto (qualquer coisa) e usa RTTI para configurar o valor da propriedade que você quiser. A propriedade pode ser qualquer uma, pois é passada como string e não depende do tipo. Resolvido meu problema, com uma linha, não dependi de tipo algum exceto TObject. O resultado foi:

```
SetPropValue(ObjetoPassado,'Foo',NovoValor);
```

Está certo que usar RTTI reduz em 5x a velocidade de um código compilado, mas pelo menos reduziu meu código em 100x, além de resolver meu problema de maneira elegante e sem apelar à POG. O que quero dizer com este exemplo é que a RTTI é um assunto tão importante, ao mesmo tempo que é tão pouco conhecido, que escolhemos como matéria de capa para esta edição especial. Confira sobre TypInfo, SetPropValue e muito mais de RTTI no excelente artigo de Gustavo Chaurais.

E temos muito mais nesta edição! Quer novidades? Que tal dicas do GridView, o MELHOR de um dos novos componentes do ASP.NET 2.0 do novíssimo RAD Studio 2007. Se quer Web, mas não em .NET, confira as 15 dicas trazidas pelo Fabrício e acabe com seus problemas no PHP. PHP também é o tema do artigo do Adriano, sobre GET e POST. Está começando? Não fique mais quieto quando o assunto for multicamadas, e aprenda com o editor como desenvolver suas primeiras aplicações distribuídas com chamadas remotas de procedimento, e também como trabalhar com a API do Windows no artigo do Adriano. E finalmente confira a conclusão do mini-curso de locadora on-line do Maikel.

Boa leitura, sucesso com o Delphi e ASP.NET 2.0!
Não deixe de conferir nosso novo BLOG: clubedelphiphp.blogspot.com

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD
Delphi Certified: 6, 7, 2005, 2006, Web, Kylix

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 600 Vídeo Aulas!
- 7 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/portal.asp* e utilize as seguintes informações: **Login: DVM.PLS e Senha: @setpropvalue**

O acesso é válido por 30 dias a partir da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Assinatura

ClubeDelphi PLUS

Mais conteúdo Delphi por menos

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+600 vídeo aulas e 7 cursos online

Brinde na web desta edição

4 Vídeos

RDPrint – Impressão de notas fiscais em impressoras matriciais-RDPrint
Nesse curso de Renato Matos você aprenderá como trabalhar com o componente RDPRINT para criação de uma classe genérica para impressão de qualquer formulário de nota fiscal. Aprenda a configurar, instalar o componente e realizar adaptações em seu programa para torná-lo o mais configurável e adaptável aos diferentes modelos de notas existentes no mercado.

Desevendando o Delphi for PHP - Parte I
http://www.devmedia.com.br/articles/viewcomp.asp?comp=8732

Desevendando o Delphi for PHP - Parte II
http://www.devmedia.com.br/articles/viewcomp.asp?comp=8787

Desevendando o Delphi for PHP - Parte III
http://www.devmedia.com.br/articles/viewcomp.asp?comp=8800

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link: *www.devmedia.com.br/clubedelphi/feedback*

Para votar, você vai precisar do código e senha de banca desta edição, que é:

Login: **DVM.PLS** Senha: **@setpropvalue**

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

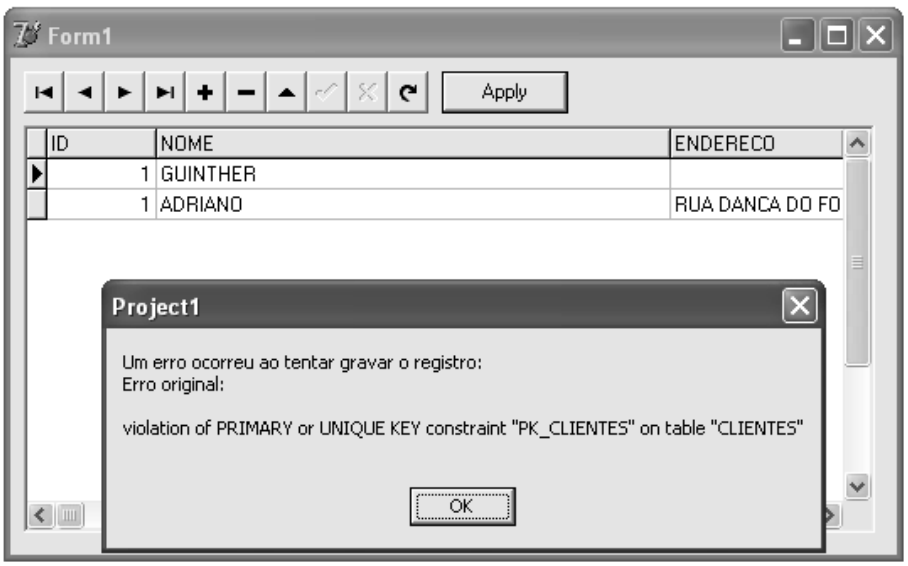
Como habilitar o Transactional Data Module no BDS 2006 e tratar erros no Delphi 7?

Por favor, me ajudem, pois estou precisando muito resolver alguns problemas. É o seguinte:

• Uso o *Delphi 2006 Studio*, porém não estou achando o *Transactional Data Module*, como posso resolver isso para ele aparecer e eu poder trabalhar?

• Usando o *Delphi 7*, desenvolvo aplicativo com o *Remote Transaction* e *ClientDataSet*, entretanto faço as ligações entre os componentes, mas na hora de gravar o registro, mesmo sendo usado o *ApplyUpdates(-1)* ou *0* ou qualquer outro parâmetro, não grava. Não dá erro, mas também não grava.

Sérgio Murilo



Respostas

Olá Sérgio Murilo, respondendo as suas perguntas, vamos lá:

• Para ativar o *Transactional Data Module* no *BDS 2006* teremos que fazer uma alteração no registro do *Windows*. Abra o registro usando *Menu Iniciar>Executar* e digite *regedit;*

• Na tela que se abre expanda a pasta *HKEY_CURRENT_USER\Software\Borland\BDS\4.0\Type Library;*

• À direita da tela clique com o botão direito do mouse e escolha *New>String value*. Dê o nome a chave de *TransactionalWizards;*

• Criada a chave clique duas vezes nela e na tela que se abre digite *True* e confirme.

Pronto isso fará com que o item procurado apareça.

Vamos a segunda pergunta. A melhor coisa se fazer é usar o evento *OnReconcileError* do *ClientDataSet*. Nesse podemos exibir o erros que ocorrem. Acesse esse evento no seu *ClientDataSet* e digite o código a seguir:

```
ShowMessage('Um erro ocorreu ao tentar gravar
            o registro:' + #13 +
           'Erro original:' + #13#13 +
           E.Message);
```

Note que estamos dando um simples *ShowMessage* e concatenando uma mensagem personalizada com a mensagem de erro (*E.Message*) capturada pelo componente. Veja na Figura um erro de *Key Violation* simulado. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

local
Data Center

Outsourcing total

...não somos uma opção!

Mudamos o conceito de que o cliente está em primeiro lugar, para nós, a qualidade dos nossos serviços está em primeiro lugar e você, nosso cliente, aparentemente está em segundo lugar, mas, será sempre o primeiro a desfrutar com satisfação a qualidade e segurança

local
Data Center

- Servidor Dedicado
- Gerenciamento
- Banda IP
- Cross Connection
- Link PAP
- Serviços compartilhados
- Sites
- E-mail
- Backup

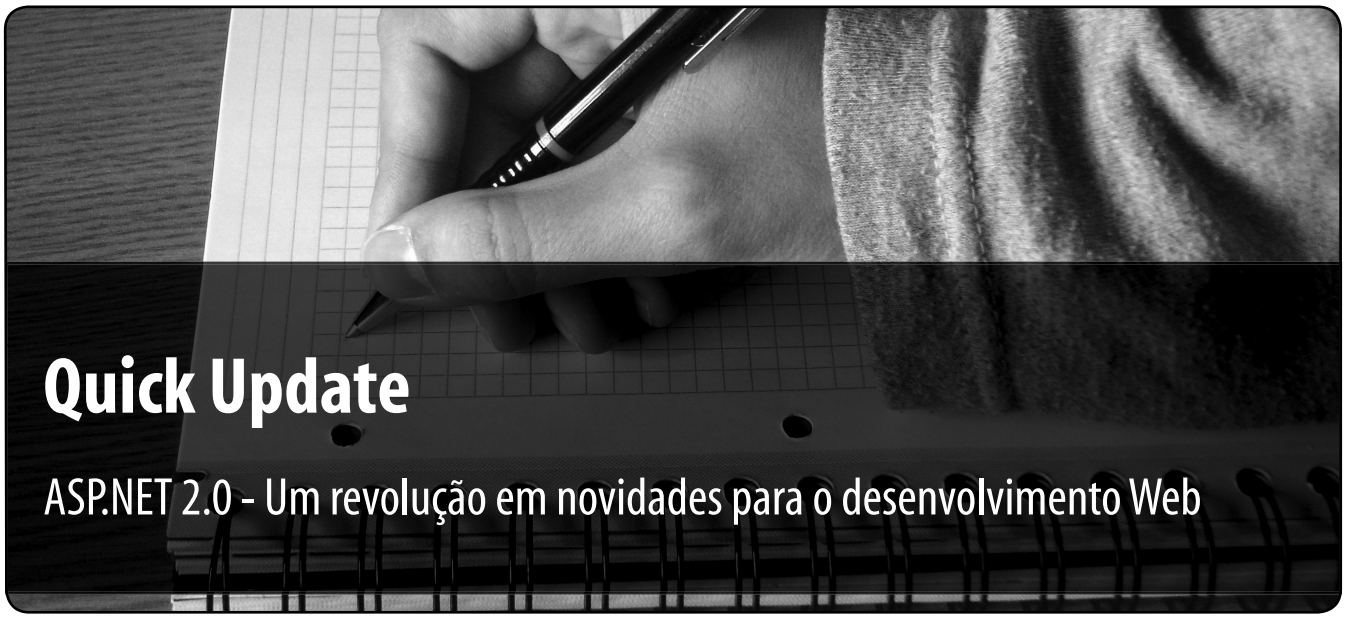
Monitoramento – Como o próprio nome diz, checar os serviços que forem destinados ao seu perfil quando solicitado de acordo com a "Entrevista de Monitoramento" prevista em contrato. Todo processo depende de uma ação pro-ativa e de mão-de-obra qualificada e ininterrupta por 24hs.

A Local tem a missão de manter por 25hs trabalhadas em uma escala de organograma e fluxograma próprio desenvolvido pelo setor de Projetos atuando com antecipação em tratamentos e prevenção de incidentes, diferenciado de mercado e criando um perfil de atendimento e regime baseado na rotina de trabalho.

3527-6510

contato@localdatacenter.com.br

Nesta seção você encontra artigos que "atualizam" o leitor rapidamente sobre um determinado assunto.



Quick Update

ASP.NET 2.0 - Um revolução em novidades para o desenvolvimento Web

Guinther Pauli
(guinther@devmedia.com.br)
é Bacharel em Sistemas de Informação, desenvolvedor Delphi há 13 anos, certificado pela Borland em Delphi: Web, 6, 7, 2005, 2006 e Kylix, e pela Microsoft: MCP, MCAD e MCSD. NET. É editor-chefe das revistas ClubeDelphi e .net Magazine. Desenvolve aplicações ASP. NET há 8 anos.

O suporte ao .NET 2.0 era uma das novidades mais esperadas pelos desenvolvedores *Delphi*. Muito disso graças ao ASP.NET 2.0, que incorpora ao desenvolvimento Web com *Delphi* uma série de novas funcionalidades. Considero a evolução do ASP.NET 1.1 para 2.0 um revolução tão grande quanto foi a do ASP clássico para o ASP.NET.

Mas o que tem de tão especial nesse ASP.NET 2.0 do *RAD Studio 2007*? Uma das principais vantagens é que ele reduz drasticamente a quantidade de código que você tem que digitar, tanto para manipulação do banco de dados, tratamento de interface, segurança e muito mais. Vamos enumerar os principais recursos e explicá-los:

DataSources – a forma como se acessa banco de dados mudou. O ADO.NET ficou "encapsulado" em novas entidades conhecidas como *DataSources*. Basicamente você larga um controle de tela, como o novo *GridView*, liga a um *DataSource* pela propriedade *DataSourceID* e pum! Tudo pronto, edição, alteração, exclusão, paginação, ordenação. Uma procura com parâmetro, com *like* por exemplo, é feita sem uma linha de código sequer. Existem vários tipos de *DataSources*, como *SqlDataSource* (para conectar ao *SQL Server*), *ObjectDataSource* (para ligar o controle a uma classe de negócio e não uma tabela, para aplicações em camadas), *XmlDataSource* e outros. Podemos dizer agora que o desenvolvimento para Web ficou RAD e *Drag* & *Drop* como é o desenvolvimento com a VCL;

Novos Controles Data-Bound – o *GridView* é o sucessor do *DataGrid* e lidera a fila dos componentes mais poderosos do ASP.NET 2.0. Ele faz tudo que o *DataGrid* fazia e mais um pouco. Totalmente personalizável. Além dele, temos os novos *FormView* e *DetailsView* para edição de um único registro por vez (você não vai mais precisar largar uma dúzia de *Text-*

Boxes e escrever código ADO.NET para inserir um registro em uma tabela).

Navegação – Temos um novo controle *TreeView*, semelhante ao da VCL, que pode ser populado com dados. Além disso, ficou muito simples criar o sistema de navegação de seu *Web Site*. Você criará um *Web Site Map* (o mapa do site, em XML), e utilizará um *Menu* ou *TreeView* para permitir a navegação. Tudo pronto, igual a uma aplicação *Desktop*. O *SiteMapPath* permitirá mostrar onde o usuário está em sua navegação. O controle *Wizard* é excelente, ele ajuda a criar uma interface paginada estilo assistente;

Master Pages – Pessoalmente acho a herança visual de formulários do *Delphi* uma das 7 maravilhas do mundo. Quando descobri que isso não existia no ASP.NET, tive um choque. Mas dava para driblar usando *User Controls* (bem melhor que o *include* do ASP clássico). No ASP.NET 2.0, temos o recursos de *Master Pages*, muito semelhante à herança visual, onde definimos uma página template e a seguir herdamos dessa página todo o conteúdo padrão, adicionando o específico;

Segurança – Novos controles de *Login* permitem que você crie todo o mecanismo de autenticação para o seu site. É só largar um *Login* na tela e pronto, temos uma interface para o usuário entrar com usuário e senha. Além disso, o próprio ASP.NET 2.0 cria um banco de dados específico para guardar usuários e outras preferências. Temos controles prontos para criação de novos usuários, recuperação de senha e mais. O controle *LoginView* permite que você defina o que cada usuário pode ver baseado no nível de segurança dele;

Themes & *Skins* – No *Windows* é possível trocar a sua aparência nas propriedades de vídeo. Todas as janelas, controles, botões, tomam uma nova forma. Podemos escolher entre vários temas pré-definidos. Agora você pode fazer o mesmo em suas aplicações Web! Que tal o usuário escolher a aparência do site? Isso tudo sem conhecer nada de estilos e CSS, tudo fácil e pronto;

Web Admin Tool – uma nova ferramenta *Web-based* permite o gerenciamento da sua aplicação Web, como configuração de usuários, grupos, privilégios, autenticação, email etc. Muitas das alterações que eram feitas no braço no *Web.Config* podem ser agora feitas através dessa ferramenta;

Sites compilados – agora é possível pré-compilar um site inteiro antes do *deploy*, inclusive os arquivos ASPX! Isso ajuda a proteger a propriedade intelectual e a segurança da aplicação, já que antes era necessário distribuir o fonte ASPX. Isso é possível através do utilitário *aspnet_compiler.exe* do *framework*;

Web Parts – Acesse *http://www.google.com/ig*. Veja que essa página contém "janelinhas" que funcionam como mini-aplicativos (mapas, previsão do tempo, notícias etc.). Experimente arrastar uma janela para outra posição, ao seu gosto, assim como está acostumado a fazer com as janelas do *Windows*. Agora você pode fazer o mesmo em suas aplicações ASP.NET 2.0, com *Web Parts*. Ideal para a construção de portais.

Cache – aprimoramentos no sistema de *Cache* também. Podemos trazer uma tabela do banco para a memória da aplicação e liberar a conexão. Os usuários ficam consultando a *cache*, e não o banco, o que aumenta a performance. Se a tabela for alterada no banco, um mecanismo de dependência automaticamente atualiza a tabela em memória evitando inconsistências;

Globalização – sabe aqueles sites que você entra e a primeira tela mostra três "bandeirinhas" com Inglês, Português e Espanhol? Você não vai precisar copiar todo o seu site depois de pronto e criar três versões traduzidas. Com o novo recurso de *Globalization* do ASP.NET 2.0, você cria seu site normalmente e depois cria arquivos de recurso para cada página, traduzindo o que for necessário (o texto de um botão por exemplo). O resto é por conta do ASP.NET 2.0. Quando o usuário entrar no site, o servidor automaticamente detecta o idioma usado pelo usuário (baseado nos cabeçalhos HTTP que o *browser* envia) e apresenta a página no seu devido idioma. Sem uma linha de código;

AJAX – O *RAD Studio 2007* incorpora agora o *framework ASP.NET AJAX*, conhecido antigamente como ATLAS. Antes no *Delphi* o AJAX era disponível somente para *IntraWeb Win32*, que foi renomeado para *VCL for the Web*. Agora sim temos um *framework* robusto e simples para o desenvolvimento com AJAX e ASP.NET. É simples porque basta você colocar um *ScriptManager* na tela, um *UpdatePanel* e dentro dele desenhar sua tela. Tudo será feito de forma assíncrona e com *updates* parciais, como chamadas XML HTTP. Tudo sem uma linha de código e configuração. Além disso, baixe o *ASP.NET AJAX Control Tool Kit* e instale no IDE. São dezenas de novos controles para AJAX que agora sim vão fazer sua aplicação se parecer com uma aplicação *Desktop*. Imagine, temos até um *PageControl* para Web. É possível também chamar um *Web Service* de forma assíncrona e receber um resultado no *client*. Mais informações e tutoriais em *ajax.asp.net*;

Melhorias gerais – controle *Substitution* parar atuar em conjunto com a *OutputCache*, *Cross-Page-Posting* para postar dados em outra página e não fazer um *postback* na própria página, controles de validação agora com grupo (para evitar de validar uma porção da tela que não tem relação com o form), melhorias no ADO.NET (caso precise usar), páginas assíncronas e muito mais.

E sabe o que é o melhor de tudo? Você verá todos, eu disse todos, esses novos recursos aqui na Revista ClubeDelphi! Então, não perca as próximas edições.

E o que vem pela frente? O ASP.NET já está na versão 3.5, com novos controles e funcionalidades. É bem provável que futuras versões do *Delphi* venham com suporte ao ASP.NET 3.5, WPF (*Windows Presentation Foundation*), WCF (*Windows Communication Foundation*), LINQ (*Language Integrated Query* – uma das maiores inovações dos últimos anos) e *SilverLight*. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

RTTI na prática

Conheça na prática um dos recursos mais poderosos da linguagem Delphi

Gustavo Chaurais
gchaurais@gmail.com
é Borland Delphi 7 Advanced Certified, Borland Delphi 2005 for Win32 Certified, Borland Delphi 2006 for Win32 Certified e Borland Delphi Instructor Certified. Foi palestrante das três últimas edições da Borland Conference Brasil e de outros grandes eventos nacionais. Além disso, é membro da coordenadoria do GIES-SC. Hoje, ministra cursos, presta consultoria e atua como Software Engineer do projeto InstallAnywhere para a empresa norte-americana Acresso.

Resumo DevMan

RTTI é um dos aspectos mais importantes da linguagem Delphi, utilizado pelo IDE e também pode ser utilizado pelo desenvolvedor, como verá neste artigo.

Muitos de nós já ouvimos falar e talvez já até tenhamos utilizado *RTTI* na prática. *Runtime Type Information*, como o próprio nome diz, são informações geradas em tempo de projeto para serem utilizadas em tempo de execução. Estas são utilizadas largamente pelos conceitos mais básicos do *Delphi*, como a listagem de propriedades no *Object Inspector*, ou a composição de um formulário em tempo de execução. Você já se perguntou como isso é feito?

Através de exemplos bastante didáticos, veremos tudo o que o *Delphi* nos disponibiliza no que se refere a *RTTI*. Prepare-se para entender como o *Delphi* funciona por trás dos panos e a utilizar tais técnicas em proveito próprio, ao projetar suas próprias aplicações.

Nesse artigo veremos
- Introdução a RTTI;
- Uso de Ponteiros;
- Criação de um simples Object Inspector para fins de didática.

Qual a finalidade
- Manipulação de propriedades em tempo de execução.

Quais situações utilizam esses recursos?
- Em sistemas orientados a objetos, podemos utilizar os recursos demonstrados para tornar ainda mais customizável o sistema para o cliente final.

Ponteiros

Antes de falarmos de *RTTI*, devemos, contudo, compreender um conceito extremamente antigo da programação de computadores: ponteiros. Com o advento da nova era das linguagens de programação (*Java*, C#, PHP, *Ruby*, etc.), tal conceito vem se tornando cada vez mais transparente e, portanto, programadores que estão ingressando na área nos dias de hoje, provavelmente,

não saberão o que realmente acontece quando um processo é executado em seu sistema operacional.

Para começar, devemos ter, pelo menos, uma vaga noção de como funciona a memória de nosso computador. Podemos resumi-la em duas regiões: *stack* e *heap*. Quando declaramos uma variável do tipo *Integer*, por exemplo, será reservada uma região do *stack* para ela. E, quando um valor for associado a essa, o processo o colocará naquela região alocada. Verifique, na **Figura 1**, como ficaria a memória após a execução do código da **Listagem 1.**

Porém, quando lidamos com objetos, por exemplo, ou outras estruturas dinâmicas (como *strings*, *arrays* dinâmicos), uma região do *stack* é novamente alocada e um ponteiro é criado para uma região alocada no *heap*. Verifique então as modificações no código (**Listagem 2**) e como ficaria, na **Figura 2**, a memória após a execução.

Resumindo, ponteiros são, basicamente, números que "apontam" para outras regiões da memória. Sim, porque, mesmo no *heap*, as alocações também são endereçadas. No exemplo anterior, é colocado, na região 2, um número significando, então, a posição da memória em que se encontra o objeto em questão.

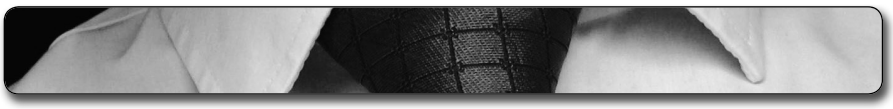
Com isso, podemos avançar um pouco e criar ponteiros um tanto mais customizados. Voltando para os tempos do Pascal, nossa regra básica será:

1. Circunflexo (^) antes = declaração;
2. Circunflexo (^) depois = leitura e associação.

Examine o código da **Listagem 3.**

Isso fará com que, no *stack*, seja alocado um espaço para um ponteiro. E, ao chamarmos *GetMem*, alocamos espaço no *heap* do tamanho de um número inteiro (32 *bits*). Ao associarmos o valor 45, este será armazenado no *heap*, e não mais no *stack*. Como fazemos com objetos, devemos liberar também os recursos alocados para nossos próprios ponteiros (*FreeMem*).

Na verdade, o *Delphi* já traz, na *unit System*, um mapeamento para ^*Integer*, o *PInteger*. Iremos utilizá-lo daqui por



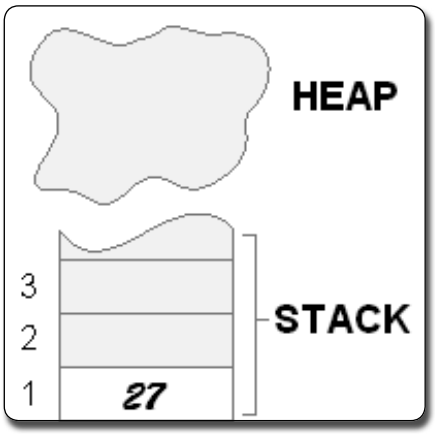


Figura 1. Representação da Memória no Primeiro Exemplo

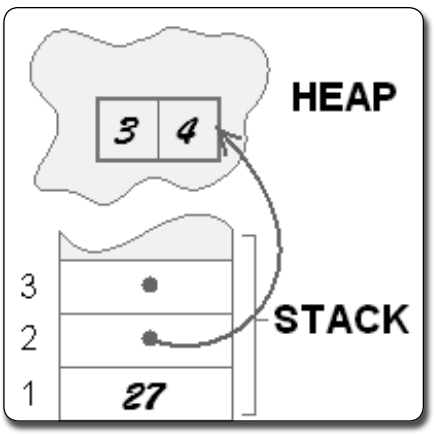


Figura 2. Representação da Memória com Ponteiros

Listagem 1. Exemplo de alocação de memória

```
procedure TForm1.Button1Click(Sender: TObject);
var
  { Alocada região 1 da memória }
  I: Integer;...
begin
..
  i := 27;
end;
```

Listagem 2. Modificação do código e nova alocação em memória

```
...
type
TTriangulo = class
public
  constructor Create(Base: Integer; Altura: Integer);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  { Região 1 alocada }
  I: Integer;
  { Região 2 alocada }
  TrianguloA: TTriangulo;
  { Região 3 alocada }
  TrianguloB: TTriangulo;
  ...
begin
  ...
  i := 27;
  TrianguloA := TTriangulo.Create(3, 4);
  TrianguloB := nil;
end;
```

Listagem 3. Código utilizando ponteiros

```
...
type
  PonteiroParaInteiro = ^Integer;
...
procedure TForm1.Button1Click(Sender: TObject);
var
  PInt: PonteiroParaInteiro;
begin
  ...
  GetMem(PInt, SizeOf(Integer));
  try
    PInt^ := 45;
    WriteLn(PInt^);
  finally
    FreeMem(PInt);
  end;
end;
```

diante. E, se você procurar por sua declaração, verá diversas outras interessantes. Seguem algumas delas:

```
PWord         = ^Word;
PDouble       = ^Double;
PVariant      = ^Variant;
PBoolean      = ^Boolean;
```

Podemos complicar mais um pouco? Veja o código da **Listagem 4** e seu resultado na **Figura 3**.

Após a visualização da figura, não fica tão difícil compreendermos o que está acontecendo. A variável *NumA*, alocada na região 1 do *stack*, aponta para uma região do *heap* alocada pelo método *GetMem*. A variável *NumB*, por sua vez, alocada na região 2 do *stack*, aponta para uma região do *heap* alocada pelo método *GetMem*, que aponta para outra região do *heap*, também alocada pelo método *GetMem*.

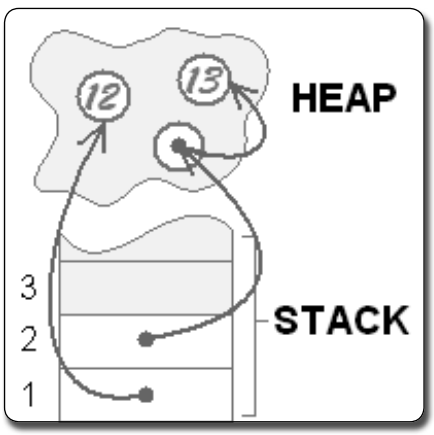


Figura 3. Representação da Memória com Ponteiros Customizados

É de fundamental importância que o conceito de ponteiros esteja bastante sólido para que não tenhamos problemas na utilização de *RTTI*. E, por mais que pareça algo bastante difícil a uma primeira visualização, saiba que, após algum treino, torna-se simples e fácil.

Para que existe RTTI?

Vamos imaginar o ambiente integrado de desenvolvimento do *Delphi*. O que são componentes? Na verdade, são, nada mais nada menos, do que código compilado inserido em um pacote, que, por sua vez, também não passa de uma simples *DLL*. Em suma, um componente não passa de um código binário, compilado, muito provavelmente, por alguma ferramenta da *Borland*/*CodeGear*.

Agora, imagine um desses compiladores transformando código fonte em binário. Quando um componente é encontrado, uma das coisas a se fazer é transformar seus elementos (atributos, métodos, propriedades, etc) em identificadores, ou seja, em tempo de execução, esses não passarão de números, endereços de memória a serem acessados e modificados.

Ok, até aqui tudo bem. A pergunta que vem a seguir seria então: se um componente no *IDE* não passa de código compilado, como o *Delphi* faz para descobrir quais são as suas propriedades e seus valores, para preencher o *Object Inspector*? O *Delphi* não possui de maneira alguma o código fonte dos componentes para consultar.

A resposta para isso é simples: *RTTI* (*Run-time Type Information*). "Informações de tipo em tempo de execução" que são adicionadas junto ao próprio código compilado, contendo diversos dados importantes sobre tipos e classes já compiladas.

Com estas informações adicionais, o *Delphi* consegue saber quais são as propriedades de um determinado componente, bem como seu nome, tipo, nome de *unit*, etc.

Você deve estar pensando: ótimo saber disso, mas existe alguma utilidade para tais informações em minhas aplicações? E a resposta é: muitas!

Se você utiliza a Orientação a Objetos a fundo em seus projetos, talvez já tenha se deparado no caso de ter vários objetos que herdam de classes completamente diferentes, porém, é necessário alterar uma propriedade do mesmo comportamento deles. Por exemplo, você está varrendo componentes para modificar sua propriedade *Caption* ou *Text*. Uma maneira de fazer isso agora seria: para cada componente, perguntar se possui *Text*. Se tiver, modifique. Caso contrário, procure por *Caption* e assim por diante.

Outra utilidade muito interessante seria na clonagem de componentes na tela. Você pode ter uma função genérica para a clonagem de qualquer componente e apenas passar para ela uma referência para o componente fonte. Ou talvez, uma aplicação customizável, na qual o usuário final teria direito a saber todas as propriedades dos componentes, modificá-las, e salvar para a próxima execução do programa.

Listagem 4. Declaração de vários ponteiros, novo exemplo

```
...
type
  PPInteger = ^PInteger;

procedure TForm1.Button1Click(Sender: TObject);
var
  { Região 1 alocada }
  NumA: PInteger;
  { Região 2 alocaca }
  NumB: PPInteger;
begin
  ...
  GetMem(NumA, SizeOf(Integer));
  GetMem(NumB, SizeOf(PInteger));
  GetMem(NumB^, SizeOf(Integer));
  try
    NumA^ := 12;
    NumB^^ := 13;
    WriteLn(NumA^);
    WriteLn(NumB^^);
  finally
    FreeMem(NumA);
    FreeMem(NumB^);
    FreeMem(NumB);
  end;
end;
```

Mãos à obra

Vamos direto a um exemplo. Isso fará com que a compreensão seja mais fácil.

No *Delphi*, crie um novo projeto e salve-o como *RTTIBasico.dpr* e o formulário principal como *Main.pas*. Adicione a este um componente *TButton*, dois *TEdits* e um outro controle visual qualquer (por exemplo, um *TLabel*, um *TCheckbox*, etc.). Nomeie este último como *controleAtual*. Opcionalmente, adicione alguns componentes para organização. O formulário

da **Figura 4** pode ser utilizado como base. Manipule o evento *OnClick* do botão para:

```
  SetPropValue(controleAtual, Edit1.Text,
Edit2.Text);
```

Adicione a *Unit TypInfo* à seção *Uses* do formulário. Rode o programa, escreva o nome de uma propriedade no primeiro *TEdit* adicionado e um valor compatível no segundo. Clique no botão e pronto. Você verá tal propriedade sendo alterada instantaneamente no componente *controleAtual*.

Como isso é possível? Simples. Ao compilar a classe do controle que você adicionou como *controleAtual*, o *Delphi* incluiu *RTTI* para sua classe. Estas informações de tipo são lidas, tanto pelo *Delphi*, na hora de preencher o *Object Inspector* e salvar o *DFM*, quanto por nosso próprio código.

Internamente, esse tipo de classe contém, além da famosa *VMT* (*Virtual Methods Table* – informações referentes a métodos virtuais), uma tabela com informações sobre *RTTI*. E, para nos ajudar, desde a sua versão 1.0, foi disponibilizada no *Delphi* uma *unit* contendo métodos para a manipulação dessas informações. Sendo que, na primeira versão, apenas sua interface nos era mostrada. Tal *unit* possui o nome de *TypInfo*.

Se você olhar com calma, verá que existem diversas funções e procedimentos com nomes bastante interessantes e, prontamente, terá vontade de aprender a utilizar todas elas! Vamos exercitar exemplos sobre a maioria.

Você irá notar a presença de dois métodos chamados *GetPropValue* e *SetPropValue*. Estes servem para lermos e configurarmos propriedades de objetos através do seu nome, respectivamente. Experimente adicionar a seguinte linha ao mesmo tratador do evento *OnClick* do botão:

```
  ShowMessage(GetPropValue(controleAtual,
Edit1.Text));
```

Ao executar tal linha, o valor atual da propriedade é mostrado na tela. Já a função *IsPublishedProp*, verifica se tal propriedade existe e é publicada em tal tipo. Adicione o seguinte código ao início do manipulador do mesmo evento:

```
  if not IsPublishedProp(controleAtual,
Edit1.Text) then
  begin
    MessageDlg(Format('Propriedade não
encontrada: %s',
    [Edit1.Text]), mtError, [mbOk], 0);
    Exit;
  end;
```

Execute o programa e digite uma propriedade inválida. A mensagem de erro é agora disparada.

Criando um Object Inspector simples

Existem alguns tipos de aplicações para a geração de relatórios nas quais é comum podermos editar relatórios e salvá-los para que o usuário final possa visualizá-lo de outras maneiras. Na verdade, qualquer aplicação do tipo *WYSIWYG* (*what you see is what you get*) seguiria este princípio. Seria disponibilizado um *designer* (como o *form designer*, ou o *ASP.NET designer* do *Delphi*) e, o que você colocar nele é o que aparecerá em tempo de execução.

Tais tipos de ferramentas, geralmente, nos disponibilizam alguma maneira de alterarmos atributos/propriedades (ou qualquer outro nome) dos elementos em questão. O *Delphi*, por exemplo, chama sua janela para tal finalidade de *Object Inspector*.

Vejamos então como poderíamos fazer pra criar um *Object Inspector* bem simples. Em seu projeto real, muito provavelmente, você adequará o exemplo a alguns componentes específicos, além de filtrar propriedades, etc.

Crie, no *Delphi*, um novo projeto com o nome *IDE.dpr* e salve o formulário principal como *Main.pas*. Adicione a ele um componente *TValueListEditor* (*ObjectInspector*) da aba *Additional*. Em seguida um *TPanel* (*pnDesigner*). Coloque, sobre o último, alguns componentes diferentes. Opcionalmente, adicione alguns componentes para organização, como um *TSplitter*. Baseie-se na **Figura 5** para montar seu formulário.

Agora vá para o código e inicie adicionando a *unit TypInfo* à seção *uses* de sua

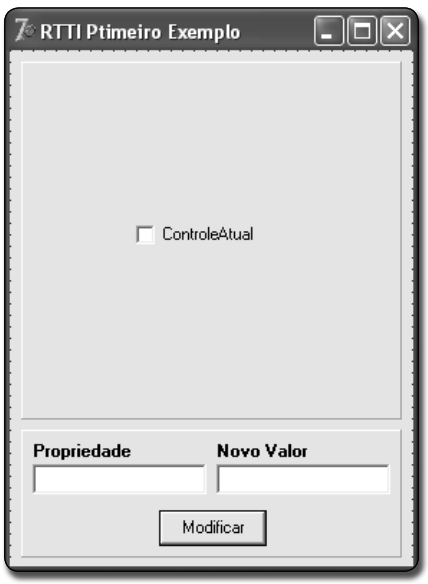


Figura 4. Formulário base para a construção do exemplo

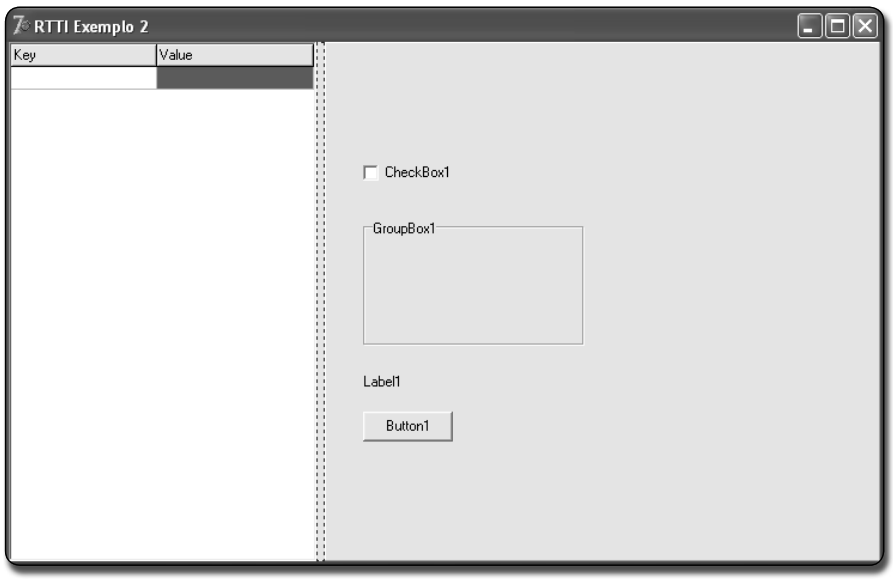


Figura 5. Formulário base para a construção do exemplo IDE

unit. Vamos começar guardando uma referência para o componente que está selecionado na tela. Portanto, na seção *private* da classe de seu formulário, adicione:

```
SelectedComponent: TComponent;
```

Declare também e implemente o seguinte método:

```
procedure TForm1.ChangeSelection(
 Component: TComponent);
  begin
  SelectedComponent := Component;
  Self.Caption := 'Editando Componente: ' +
    Component.Name;
end;
```

Agora, selecione todos os componentes que estão sobre o *pnDesigner* e manipule o evento *OnMouseDown* de todos para o mesmo método:

```
ChangeSelection(Sender as TComponent);
```

O evento *OnCreate* do formulário deve ser manipulado também para selecionar um dos componentes adicionados sobre *pnDesigner* (de sua escolha):

```
  ChangeSelection(CheckBox1);
```

Neste instante, se você executar sua aplicação e clicar sobre os componentes do *pnDesigner*, verá que o texto da barra de títulos muda de acordo. Agora, vamos declarar e implementar um dos principais métodos de nosso exemplo. Declare uma nova *procedure* com o nome *RefreshProperties* e implemente como mostrado na **Listagem 5**.

Este será o método responsável por preencher as propriedades no *Object Inspector*. Vamos estudá-lo com cuidado. Primeiramente, chamamos o método *GetPropList*. Em seu primeiro parâmetro, nos é solicitada uma estrutura do tipo *PTypeInfo*, ou seja, um ponteiro para a tabela de *RTTI* de uma classe ou tipo. Existem duas maneiras bastante conhecidas de recuperarmos este ponteiro: a primeira é utilizando o retorno da função *ClassInfo* de qualquer classe; a segunda é utilizando a função *TypeInfo* e passando um tipo como parâmetro. As duas vão retornar um ponteiro exatamente para a região da memória onde está alocada a tabela de *RTTI* daquele tipo. Porém, convém sabermos a diferença entre as duas: a segunda é um tipo especial de método que também podemos chamar de *pseudo-função* (como *Lenght*, *High*, *Low*, *Pred*, etc.), o que significa ser gerada pelo compilador em *compile time* (o compilador simplesmente adiciona o código naquele ponto, em vez de colocar uma chamada a uma função existente). Já a primeira é executada totalmente em *run-time*, ou seja, existe um método real a ser chamado para tal tarefa. Em suma, *TypeInfo* acaba se tornando ligeiramente mais rápido que *ClassInfo*. Contudo, você deve escolher a que se adaptar melhor a cada caso.

GetPropList, aceita, além de uma estrutura *PTypeInfo*, um filtro de propriedades e um ponteiro para uma lista de *TPropInfos* (*PPropList*) como parâmetro. O retorno da função é o número de propriedades *published* que tal tipo possui. E, como sabemos que *TPropList* é uma lista de *TPropInfos*, chamamos *GetMem* para alocar esta lista com o tamanho correto (número de propriedades multiplicado pelo tamanho de cada *TPropInfo*).

Feito isso, vamos chamar uma versão sobrecarregada (*overloaded*) e simplificada de *GetPropList*, que pode aceitar diretamente um *TObject* e uma lista *PPropList* como parâmetros. Isso irá preencher nossa lista (já alocada na memória) com todas as propriedades do tipo do componente selecionado.

Após, recuperamos a estrutura *TPropInfo* de cada propriedade e chamamos *GetPropValue* para pegar seu valor. São vários os dados que podemos obter das propriedades (como *Name*, que acabamos de utilizar). Na **Listagem 6** podemos ver a declaração da estrutura *TPropInfo* (*TypInfo.pas*).

Note também algo bastante interessante: para pegar a estrutura *TPropInfo*, utilizamos *TPropInfo(PropList^[i]^)*. Isso porque, em *PropList*, temos um ponteiro para uma lista (por isso, o "^" após *PropList*). E, ao acharmos o elemento na lista, teremos um ponteiro para um *TPropInfo* (por isso, o segundo "^").

Agora, adicione a seguinte chamada ao final do seu método *ChangeSelection*:

```
  RefreshProperties;
```

Listagem 5. Código para atualizar as propriedades

```
procedure TForm1.RefreshProperties;
var
  PropList: PPropList;
  PropCount: Integer;
  i: Integer;
  PropInfo: TPropInfo;
  PropValue: Variant;
begin
  ObjectInspector.Strings.Clear;

  PropCount := GetPropList(SelectedComponent.ClassInfo, tkAny, nil);
  GetMem(PropList, PropCount * SizeOf(TPropInfo));
  try
    GetPropList(SelectedComponent, PropList);
    SortPropList(PropList, PropCount);

    for i := 0 to Pred(PropCount) do
    begin
      PropInfo := TPropInfo(PropList^[i]^);
      PropValue := GetPropValue(SelectedComponent, PropInfo.Name);
      ObjectInspector.InsertRow(PropInfo.Name, VarToStr(PropValue), True);
    end;
  finally
    FreeMem(PropList);
  end;
end;
```

Listagem 6. Declaração da estrutura de TPropInfo

```
  TPropInfo = packed record
    PropType: PPTypeInfo;
    GetProc: Pointer;
    SetProc: Pointer;
    StoredProc: Pointer;
    Index: Integer;
    Default: Longint;
    NameIndex: SmallInt;
    Name: ShortString;
  end;
```

Execute o programa e clique sobre os componentes. Você verá que o *Object Inspector* é preenchido com as propriedades de cada um corretamente.

Para finalizar esta parte do exemplo, vamos adicionar edição a ele. Simplesmente manipule o evento *OnKeyPress* de *ObjectInspector* usando a **Listagem 7**.

A explicação deste é bastante simples. Simplesmente recuperamos o nome e o valor da propriedade e chamamos *SetPropValue* passando o componente selecionado. Teste sua alteração editando e pressionando o *Enter* sobre as propriedades do *ObjectInspector*.

Vamos incrementar mais um pouco adicionando um recurso de remoção de componentes. Declare e implemente o seguinte método com o nome *DeleteComponent*. O código está descrito na **Listagem 8**.

Apenas estamos removendo o controle de seu *parent* e liberando-o. Caso o componente *deletado* seja o componente atual, mudamos a seleção para o primeiro controle de *pnDesigner*.

Modifique o evento *MouseDown* dos componentes, configurado anteriormente, para:

```
  if ([ssCtrl, ssShift] * Shift) = [ssCtrl,
ssShift] then
    DeleteComponent(Sender as TComponent)
  else
    ChangeSelection(Sender as TComponent);
```

Ou seja, se as teclas *Ctrl* e *Shift* estiverem pressionadas ao clicarmos no componente, este será removido. Faça o teste executando seu programa.

Personalizando ainda mais o exemplo

Já que estamos permitindo tamanha alteração nos componentes de nosso *designer*, porque não facilitar a vida do usuário, adicionando mais algumas funções ao clique do *mouse*? Existem algumas mensagens da *API* do *Windows* muito interessantes, que vão combinar perfeitamente com este caso. Declare e implemente a função descrita na **Listagem 9.** Troque também o evento *MouseDown* dos componentes para o código da **Listagem 10**.

Execute seu programa e veja o que aconteceu. Se você clicar bem no meio de um componente, poderá arrastá-lo pelo

Listagem 7. Manipulação do evento OnKeyPress do nosso Object Inspector

```
procedure TForm1.ObjectInspectorKeyPress(Sender:
  TObject; var Key: Char);
var
  PropertyName: string;
  PropertyValue: string;
begin
  if Key = #13 then
  begin
    PropertyName := ObjectInspector.
      Keys[ObjectInspector.Selection.Top];
    PropertyValue := ObjectInspector.
      Values[PropertyName];
    SetPropValue(SelectedComponent,
      PropertyName, PropertyValue);
  end;
end;
```

Listagem 8. Código para apagar componentes

```
procedure TForm1.DeleteComponent(Component:
  TComponent);
var
  SelectFirst: Boolean;
begin
  if pnDesigner.ControlCount > 1 then
  begin
    SelectFirst := SelectedComponent = Component;
    pnDesigner.RemoveControl(Component as TControl);
    FreeAndNil(Component);
    if SelectFirst then
      ChangeSelection(pnDesigner.Controls[0]);
  end;
end;
```

Listagem 9. Novos recursos no designer

```
function TForm1.GetSyscommandMessage(Sender:
  TControl; X, Y: Integer): Integer;
var
  ParteVertical: Integer;
  ParteHorizontal: Integer;
begin
  ParteVertical := Sender.Height div 3;
  ParteHorizontal := Sender.Width div 3;
  if X < (ParteHorizontal) then
  begin
    if Y < (ParteVertical) then
      { TopLeft }
      Result := $F004
    else if Y > (Sender.Height - ParteVertical) then
      { BottomLeft }
      Result := $F007
    else
      { Left }
      Result := $F001;
  end
  else if X > (Sender.Width - ParteHorizontal) then
  begin
    if Y < (ParteVertical) then
{ TopRight }
      Result := $F005
    else if Y > (Sender.Height - ParteVertical) then
{ BottomRight }
      Result := $F008
    else
{ Right }
      Result := $F002;
  end
  else
  begin
    if Y < (ParteVertical) then
 { Top }
      Result := $F003
    else if Y > (Sender.Height - ParteVertical) then
{ Bottom }
      Result := $F006
    else
{ Center }
      Result := $F012;
  end;
end;
```

Listagem 10. Modificação do evento MouseDown para mover objetos

```
procedure TForm1.CheckBox1MouseDown(Sender:
  TObject; Button: TMouseButton; Shift:
  TShiftState; X, Y: Integer);
begin
  if ([ssCtrl, ssShift] * Shift) = [ssCtrl, ssShift] then
    DeleteComponent(Sender as TComponent)
  else
  begin
    ChangeSelection(Sender as TComponent);
    ReleaseCapture;
    PostMessage((Sender as TWinControl).Handle,
      WM_SYSCOMMAND, GetSyscommandMessage((Sender as TControl), X, Y), 0);
  end;
end;
```

Listagem 11. Clonagem de componentes

```
function TForm1.CloneComponent(Component:
  TComponent): TComponent;
var
  PropList: PPropList;
  PropCount: Integer;
  i: Integer;
  PropInfo: TPropInfo;
  PropValue: Variant;
  PropTypeKind: TTypeKind;
  TmpObjProp: TObject;
  TmpIntfProp: IInterface;
  TmpMethodProp: TMethod;
  ComponentClass: TComponentClass;
  ResultName: string;
begin
  SelectedComponent := Component;
  ComponentClass := TComponentClass(
    Component.ClassType);
  Result := ComponentClass.Create(Component.Owner);
  i := 1;
  repeat
    ResultName := 'Clonado' + IntToStr(i);
    Inc(i);
  until FindComponent(ResultName) = nil;
  PropCount := GetPropList(SelectedComponent
    .ClassInfo, tkAny, nil);
  GetMem(PropList, PropCount * SizeOf(TPropInfo));
  try
    GetPropList(SelectedComponent.ClassInfo, PropList);
    for i := 0 to Pred(PropCount) do
    begin
      PropInfo := TPropInfo(PropList^[i]^);
      PropValue := GetPropValue(SelectedComponent,
        PropInfo.Name);
      PropTypeKind := PropType(SelectedComponent,
        PropInfo.Name);
      if PropTypeKind = tkClass then
      begin
        TmpObjProp := GetObjectProp(SelectedComponent,
          PropInfo.Name);
        SetObjectProp(Result, PropInfo.Name, TmpObjProp);
      end
      else if PropTypeKind = tkInterface then
      begin
        TmpIntfProp := GetInterfaceProp(
          SelectedComponent, PropInfo.Name);
        SetInterfaceProp(Result, PropInfo.Name, TmpIntfProp);
      end
      else if PropTypeKind = tkMethod then
      begin
        TmpMethodProp := GetMethodProp(
          SelectedComponent, PropInfo.Name);
        SetMethodProp(Result, PropInfo.Name, TmpMethodProp);
      end
      else
      begin
        if PropInfo.Name = 'Name' then
          Result.Name := ResultName
        else if PropInfo.Name = 'Left' then
          (Result as TControl).Left := (Component as TControl).Left + 15
        else if PropInfo.Name = 'Top' then
          (Result as TControl).Top := (Component as TControl).Top + 15
        else
          SetPropValue(Result, PropInfo.Name,
            PropValue);
      end;
    end;
  finally
    FreeMem(PropList);
  end;
  (Result as TControl).Parent := (Component as TControl).Parent;
end;
```

pnDesigner. Se clicar mais para um dos cantos, poderá redimensioná-lo.

Isso é possível porque as mensagens do tipo *WM_SYSCOMMAND*, quando enviadas aos controles, permitem diversas alterações com o próprio *mouse*. Porém, só irão funcionar com componentes que derivam de *TWinControl*, pois necessitam de um *Handle* válido do *Windows*.

O número da mensagem é determinado na função *GetSyscommandMessage*. É ele que determinará como o controle deve se comportar ao movimento do *mouse*. O método é bastante auto-explicativo.

Por que não salvar janelas e deixar o usuário final organizá-las da maneira que preferir? Você poderia implementar isto em seu programa e deixá-lo bastante flexível, para que qualquer usuário possa deixá-lo mais fácil de ser operado. Alguns setores podem querer que determinados controles fiquem em tal posição, diferentemente de outros, que podem preferir outras configurações. Fica então como treino e matéria para próximos artigos, salvar e restaurar as propriedades dos componentes.

Clonando componentes

Existem diversas situações nas quais precisamos criar componentes dinamicamente no formulário para que o usuário possa entrar com informações de maneira fácil. Porém, copiar propriedades de um componente para outro na hora de criar clones na tela é uma tarefa bastante "chata".

Vejamos como poderíamos construir uma função genérica de clonagem, para a qual, passaríamos apenas um componente e tudo estaria pronto. Declare e implemente a função da **Listagem 11** a seguir e vamos estudá-la:

A primeira coisa a ser feita é criar um objeto da mesma classe (*ClassType*) do componente original. O *Owner* também deverá ser o mesmo. Após isso, tentamos encontrar um nome para o novo componente. Para isso tentamos *Clonado1* e, se este já existir, incrementamos o último número até que um nome disponível seja encontrado.

A técnica de ler todas as propriedades de determinado componente já foi explicada

anteriormente, na parte do preenchimento do *Object Inspector*, então, utilizaremos, basicamente, o mesmo código.

A parte mais complicada deste método se encontra no seguinte problema: alguns tipos de propriedades não podem ser convertidos para *Variant*, portanto, não podem ser lidas e escritas por *GetPropValue* e *SetPropValue*. Desta vez, iremos utilizar alguns métodos mais específicos para trabalhar com estes tipos.

Para ler o tipo da propriedade, uma vez que tenhamos a estrutura *TPropInfo* em mãos, podemos chamar o método *PropType*, que nos retornará um valor *TPropKind*. A partir deste, se estivermos lidando com uma propriedade do tipo *tkClass*, *tkInterface* ou *tkMethod*, em vez de *GetPropValue*, chamaremos *GetObjectProp*, *GetInterfaceProp* e *GetMethodProp*, respectivamente. Bem como *SetObjectProp*, *SetInterfaceProp* e *SetMethodProp* em vez de *SetPropValue*.

Para os outros tipos de propriedade, apenas chamamos *SetPropValue* e já está tudo ok. No entanto, algumas outras propriedades ainda precisam de tratamento especial. *Name* receberá o nome calculado anteriormente. *Left* e *Top* são acrescidos de 15 para evitar que um componente seja posicionado sobre o outro, escondendo o original.

Finalizando, configuramos o mesmo *Parent* para os dois componentes e a função de clonagem está pronta. Precisamos apenas, modificar novamente o evento *OnMouseDown* dos componentes para a codificação da **Listagem 12**.

Faça um teste clicando sobre um componente com as teclas *Ctrl* e *Alt* pressionadas. Você verá que uma cópia do componente é criada logo ao lado e abaixo.

Difícil? Um pouco. Porém, mais uma vez, treino e prática farão esta atividade rotineira e fácil.

RTTI para minhas próprias classes

Vamos dizer que você está desenvolvendo o sistema de *logs* de seu projeto e, este, é baseado em *RTTIs* de objetos. Você teria um objeto ou uma função para cadastrar uma informação de *log* e esta arquivaria basicamente propriedades.

Crie um novo projeto chamado *Logs.dpr* e salve o formulário principal como *Main.pas* e adicione ao formulário: dois *TButtons* e um *TCheckbox*. Declare *TypInfo* na sua seção *Uses*. Modifique o *Caption* dos botões para "Logar componente" e "Logar objeto". Baseie seu formulário na **Figura 6**.

No código, declare e implemente a função da **Listagem 13**. Note que estamos basicamente lendo todas as propriedades do objeto, armazenando-as em um *stream*, e mostrando-as, através de um *ShowMessage*.

Manipule o evento *OnClick* do botão com a finalidade de "logar componente" para:

```
Log(CheckBox1);
```

Ao executar e clicar no botão, a mensagem aparecerá na tela com todas as propriedades do componente. Aqui, você modificaria para o seu próprio sistema de *log* (arquivo, banco de dados, etc.).

Agora, digamos que você queira possibilitar o *log* de seus próprios objetos também. Declare e implemente então a classe *TContaBancaria* como segue na **Listagem 14**. E por fim, manipule o evento *OnClick* do segundo botão conforme a **Listagem 15**.

Figura 6. Formulário base para a construção do exemplo de Logs

Listagem 12. Modificação do evento MouseDown

```
procedure TForm1.CheckBox1MouseDown(Sender:
  TObject; Button: TMouseButton; Shift:
  TShiftState; X, Y: Integer);
begin
  if ([ssCtrl, ssAlt] * Shift) = [ssCtrl, ssAlt] then
    ChangeSelection(CloneComponent(Sender as TComponent))
  else if ([ssCtrl, ssShift] * Shift) = [ssCtrl, ssShift] then
    DeleteComponent(Sender as TComponent)
  else
  begin
    ChangeSelection(Sender as TComponent);

    ReleaseCapture;
    PostMessage((Sender as TWinControl).Handle, WM_SYSCOMMAND,
      GetSyscommandMessage((Sender as TControl), X, Y), 0);
  end;
end;
```

Listagem 13. Código do Log Classe

```
procedure TForm1.Log(const Objeto: TObject);
var
  PropList: PPropList;
  PropCount: Integer;
  i: Integer;
  PropInfo: TPropInfo;
  PropValue: Variant;
  Str: string;
begin
  Str := '';

  PropCount := GetPropList(Objeto.ClassInfo,
    tkAny, nil);
  GetMem(PropList, PropCount * SizeOf(TPropInfo));
  try
    GetPropList(Objeto, PropList);
    SortPropList(PropList, PropCount);

    for i := 0 to Pred(PropCount) do
    begin
      PropInfo := TPropInfo(PropList^[i]^);
      PropValue := GetPropValue(Objeto,
        PropInfo.Name);
      Str := Str + PropInfo.Name + ': ' +
        VarToStr(PropValue) + sLineBreak;
    end;
  finally
    FreeMem(PropList);
  end;

  ShowMessage(Str);
end;
```

Tente executar e clicar no botão. Você provavelmente ficará desapontado e talvez pensando que fez algo errado. Todavia, o *Access Violation* disparado é proposital.

Para que seja gerado *RTTI* para alguma classe, uma condição deve ser atendida: é preciso que o compilador saiba que deve gerar tais informações. Para tanto, temos que compilar a classe com a diretiva *{$TYPEINFO ON/OFF}*.

Faça isso. Coloque *{$TYPEINFO ON}* e *{$TYPEINFO OFF}* antes e após a declaração de sua classe, respectivamente. Agora teste o programa e, finalmente, as informações aparecerão no *ShowMessage*!

Informações de *RTTI* são geradas apenas para elementos da seção *published* de sua classe. Esse é o porquê de termos de colocar as propriedades de nossos componentes nesta seção para que apareçam no *Object Inspector*. Além disso, em vez de *{$TYPEINFO ON/OFF}*, muito comumente utilizamos suas contrações *{$M+}* e *{$M-}*.

Classes que descendam de outras classes que tenham sido compiladas com esta diretiva gerarão *RTTI* automaticamente. Esta é, na verdade, a única razão da existência da classe *TPersistent* na *VCL*, da qual todo componente herda e que é compilada desta maneira. Se você desejar então, retire as diretivas e apenas modifique sua classe para que descenda de *TPersistent*. O mesmo acontecerá.

Onde mais encontro RTTI

Na verdade, no *Delphi*, existe muito mais *RTTI* do que nos é disponibilizado em *TypInfo*. Por exemplo, operadores *is* e *as* utilizam tais informações para funcionarem completamente. Além disso, tipos básicos como *enumerations* também geram seu próprio *RTTI*.

Para fazer um pequeno teste, chame a função *GetEnumName* (*TypInfo*) para um tipo enumerado, como segue abaixo:

```
ShowMessage(GetEnumName(TypeInfo(TFormStyl
e), Integer(Self.FormStyle)));
```

Isso mostraria o *string* correspondente ao *FormStyle* do formulário atual (por exemplo: *'fsNormal'*). E, mais uma vez, isso só é possível devido ao *RTTI*. Se este não existisse, não teríamos estas informações em tempo de execução. Teríamos apenas números representando tais valores. E isto é também largamente utilizado pelo *Object Inspector* e pelo *Form Designer* para mostrar nomes correspondentes aos valores de *enumerations* ao usuário.

Podemos encontrar *RTTI* também em métodos e campos *published*, porém recuperar estas informações é uma tarefa bastante difícil e não suportada abertamente pela *Borland/CodeGear*. Não há uma *unit* como *TypInfo* para tanto. A partir do *Delphi 7* temos ainda a diretiva *{$METHODINFO ON/OFF}* que nos possibilita verificar informações inclusive de métodos publicados (*public*). Caso esteja interessado em tais informações, verifique o *blog* de *Hallvard Vassbotn Oslo http://hallvards.blogspot.com*. Lá você irá encontrar diversos *hacks* (técnicas não oficiais) sobre como obter esse tipo de *RTTI*. *Units* referentes à parte de *SOAP* e *WebServices* do *Delphi* também contêm bastante código pertinente.

Conclusão

Como pudemos observar, o *RTTI* está por trás dos procedimentos mais comuns do *IDE* do *Delphi*. Podemos então tirar proveito disso em nossas próprias aplicações, já que a *unit TypInfo* foi disponibilizada para tal fim.

Uma dica final: sempre prefira utilizar a Orientação a Objeto em detrimento de técnicas como esta. Pois, além de complicar a legibilidade do código, você estará quebrando a cadeia de flexibilidade proporcionada pelo paradigma O.O.. Treine bastante e aprenda outras técnicas como essa para deixar seus projetos sempre com a maior legibilidade e otimização possível. ●

Listagem 14. Declaração e implementação da classe TContaBancaria

```
...
  TContaBancaria = class
  private
    FBanco, FAgencia, FNumero: Integer;
  public
    constructor Create(Banco, Agencia, Numero: Integer);
  published
    property Banco: Integer read FBanco;
    property Agencia: Integer read FAgencia;
    property Numero: Integer read FNumero;
  end;
...
implementation
...
constructor TContaBancaria.Create(Banco, Agencia, Numero: Integer);
begin
  FBanco := Banco;
  FAgencia := Agencia;
  FNumero := Numero;
end;
```

Listagem 15. Botão Log Objeto

```
procedure TForm1.Button2Click(Sender: TObject);
var
  ContaBancaria: TContaBancaria;
begin
  ContaBancaria := TContaBancaria.Create(333,
   4444, 55555);
  try
    Log(ContaBancaria);
  finally
    ContaBancaria.Free;
  end;
end;
```


Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

POR QUE A ALOG
É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

GridView no RAD 2007

Dicas de como usar o Grid turbinado do ASP.NET 2.0

Resumo DevMan

O GridView é o componente mais importante do ASP.NET 2.0, utilizado para mostrar dados em forma de Grid. Aprenda neste artigo várias dicas de como manipular este poderoso componente.

Nesse artigo veremos

- Principais dicas sobre o controle GridView do ASP.NET 2.0;
- Visualização de imagens no GridView;
- Rodapé com totalização;
- Master/Detail e Master/Detail Template.

Qual a finalidade

- Visualizar e conhecer as principais novidades do GridView.

Quais situações utilizam esses recursos?

- Em todo tipo de web site é possível usar o GridView, pois é um componente excelente para visualizar e editar dados.

Luciano Pimenta
lucianoalmeidapimenta@gmail.com
é Técnico em Processamento de Dados, desenvolvedor Delphi. Palestrante da 4ª edição da Borland Conference (BorCon).
Blog: http://lucianopimenta.blogspot.com.

Desde a minha primeira experiência com o ASP.NET, um dos controles que mais utilizei foi o *Grid*, pela sua facilidade em mostrar os dados no formato de tabela. Primeiramente, usei o *DataGrid* da versão 1.x do *.NET Framework* e agora uso *GridView* para o .NET 2.0 no *RAD Studio 2007*.

A primeira idéia de quem começa com ASP.NET é fazer aplicações Web iguais a aplicações *Desktop*, mas claro, nem tudo é possível. Especificamente, em relação ao *Grid*, adicionar imagens, controles e agrupamento, são as primeiras funcionalidades que desejamos.

O *DataGrid* é muito customizável, mas o *GridView* trouxe muito mais facilidades ao desenvolvedor, como inserção, atualização, exclusão e seleção de dados, de forma muito simples. Neste artigo, veremos algumas dicas úteis para a utilização desse controle para customizá-lo em suas aplicações Web.

Construindo o exemplo e adicionando controles

A funcionalidade de *templates* do ASP.NET ajuda em muito a customização do *GridView* para a adição de controles. Basta criar um coluna do

controle do tipo *TemplateField* e adicionar os controles necessários.

Neste exemplo, vamos trabalhar com o *CheckBox*, onde simularemos um *WebMail*. Para quem está acostumado a usar *WebMail*, sabe que os e-mails aparecem em um *Grid*, onde podemos selecioná-los, excluí-los entre outras funcionalidades.

Crie uma nova aplicação Web no *RAD Studio 2007*, com suporte ao AJAX (**Figura 1**), através do *menu File>New>Other>Delphi for .NET Projects>AJAX-Enabled ASP.NET Web Application*. Usaremos AJAX para que algumas dicas fiquem de fácil entendimento, já que não teremos *refresh* total da página.

Conexão com o banco

Adicione um *GridView* (aba *Web Data*) dentro do *UpdatePanel* na página e acesse a *Smart Tasks* do mesmo, acessando *New Data source* em *Choose Data Source*. Na tela que se abre, escolha a opção *Database* e clique em OK (**Figura 2**).

ClubeDelphi PLUS www.devmedia.com.br/dubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como desenvolver uma aplicação Web com Ajax no RAD Studio 2007

*http://www.devmedia.com.br/articles/viewcomp.asp?comp=8357&hl=*asp.net*%20and%20*2.0**

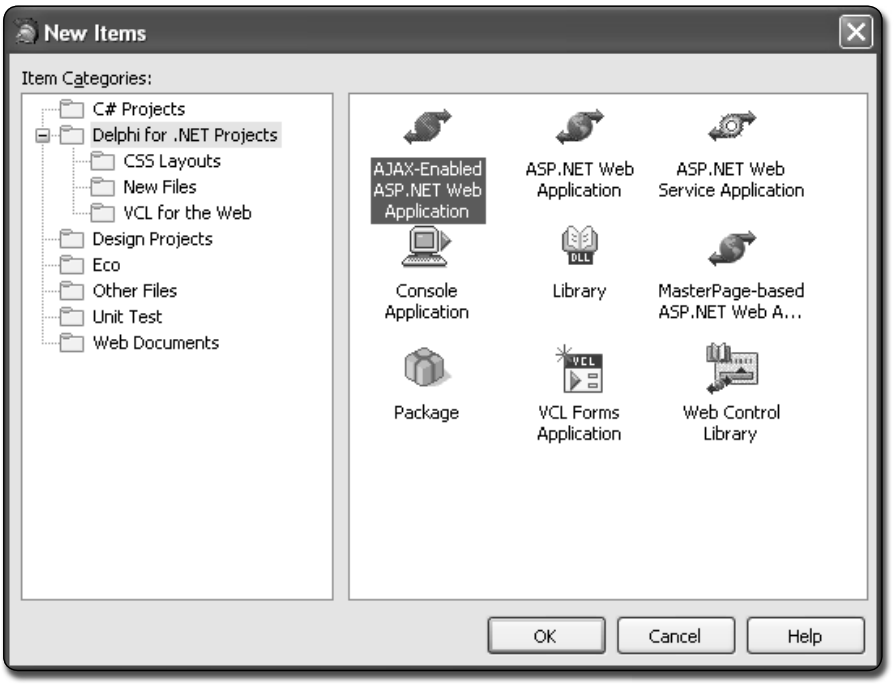


Figura 1. Criando o projeto Web com suporte ao AJAX

Nota do DevMan

O AJAX (Assyncronous JavaScript and XML) já era suportado pelo Delphi na versão 2007 Win32, com o IntraWeb, renomeado para VCL Web por questões de marketing. Mas o poder todo da tecnologia veio à tona só agora, com o framework ASP.NET AJAX da MS (codename ATLAS), incorporado nativamente no IDE do RAD Studio 2007. Aplicações que fazem uso do ASP.AJAX, além de entrarem na nova era da Web 2.0, são também conhecidas como RIA (Rich Internet Applications), onde o usuário tem uma experiência muito maior com a interface do que tradicionais aplicações Web. Aplicações Web são agora semelhantes à aplicações Desktop, pelo fato de não haver um refresh total da página (branco) quando é efetuado um postback (clique em um botão por exemplo). O AJAX se encarrega de atualizar somente porções da tela, ao invés de todo o Web Form. As requisições são assíncronas, não travam o cliente e são feitas não diretamente pelo form da página, mas através de uma chamada paralela em XML HTTP, protocolo suportado hoje pelas versões mais atuais dos Web Browsers. Outro exemplo de aplicação RIA são as construídas com a nova tecnologia do .NET chamada Silverlight, algo como é o Flash, que provavelmente será suportado em futuras versões do Delphi.

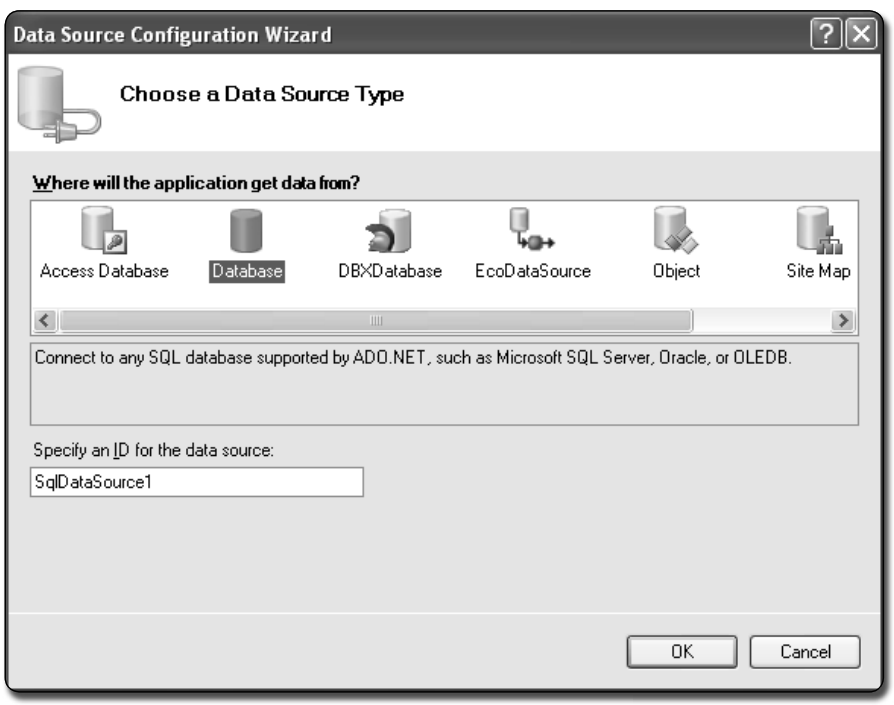


Figura 2. Escolhendo o Data Source para conexão com o banco

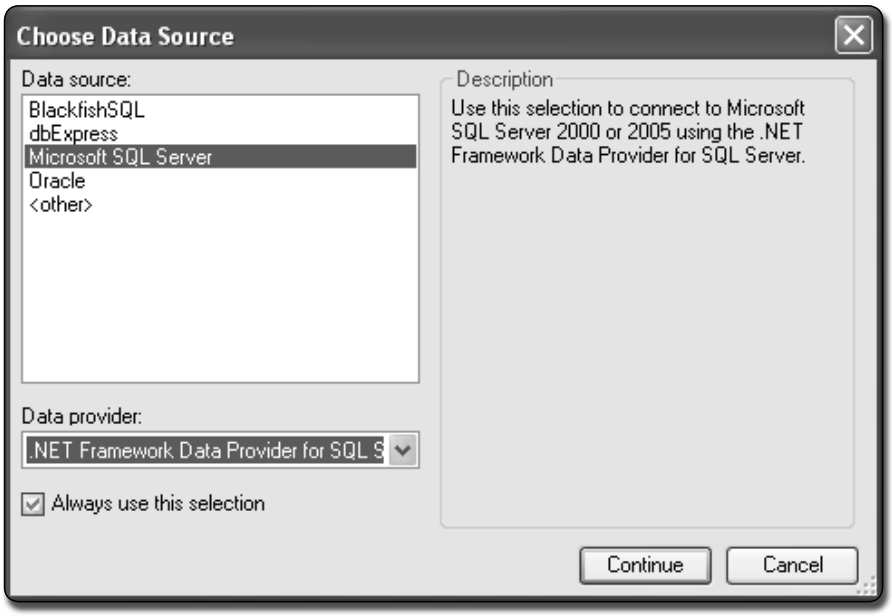


Figura 3. Criando a conexão com o banco

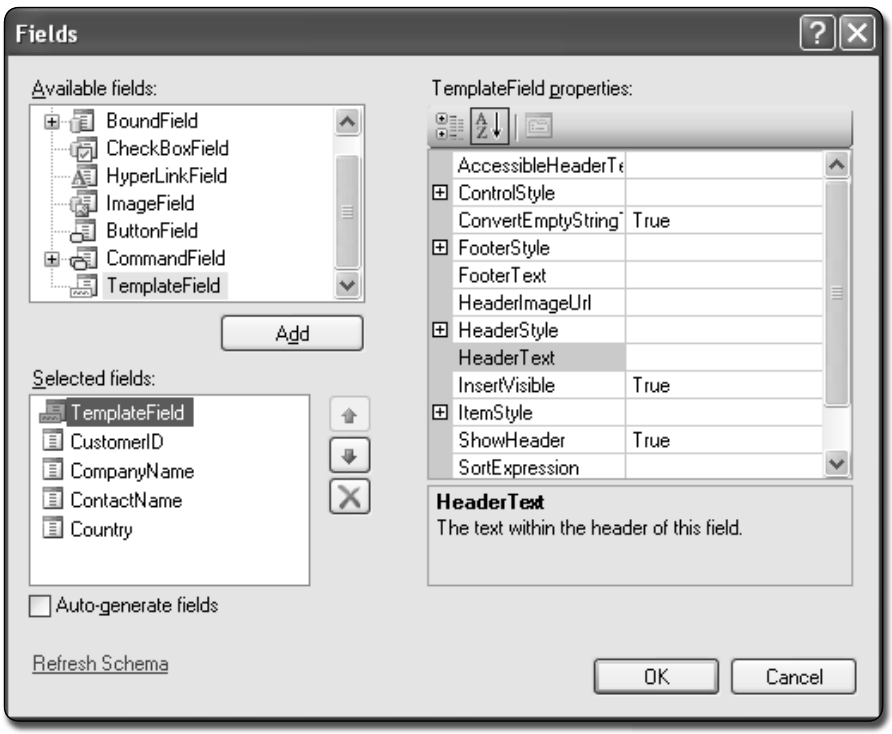


Figura 4. Adicionando um TemplateField no GridView

Crie uma conexão com o banco de dados (neste exemplo, usaremos *SQL Server 2005*), clicando em *New Connection* e será aberta uma outra tela, onde você deve escolher em *Data source* a opção *Microsoft SQL Server* e em *Data provider* selecione o *provider* para *SQL Server* (**Figura 3**).

Clique em *Continue* e escolha o servidor e o banco de dados *Northwind*. Você pode usar outro banco de sua preferência. Clique em *Next* no *wizard* até chegar a opção de escolha das tabelas do banco.

Para este exemplo, use os dados da tabela *Customer* (selecione apenas alguns campos). Clique em *Next* e *Finish* para terminar a consulta ao banco de dados. Acesse a opção *Auto Format* nas *tasks* e escolha *Professional*.

TemplateField

Deixe o controle com a opção de paginação ativada (propriedade *AllowPaging*). Acesse a propriedade *Columns* do *GridView*, adicione um campo do tipo *TemplateField* e clique em OK (**Figura 4**).

Acesse a *Smart Tasks* do *GridView* e clique na opção *Edit Templates*. Na seção *ItemTemplate*, adicione um *CheckBox*. Feche o editor, rode a aplicação e veja os *CheckBoxes* adicionados ao *GridView* (**Figura 5**).

SQL Server, Northwind e Magement Studio Express

Neste artigo, utilizaremos o SQL Server 2005 Express, free, comprovadamente o MELHOR banco para se trabalhar com aplicações ASP.NET. Para gerenciar o banco de dados, utilizaremos uma ferramenta chamada SQL Server Management Studio Express (semelhante ao que é o IBExpert para o Firebird). Você precisa primeiro instalar o SQL Server, depois o Management, e a seguir registrar o banco Northwind clicando de direita no ícone Databases>Attach. O downloads estão listados abaixo, a instalação é muito simples e não detalharei aqui.

SQL Server 2005 Express
http://www.microsoft.com/express/download/

SQL Server 2005 Express e Management Studio Express
http://msdn.microsoft.com/vstudio/express/sql

SQL Server Samples Databases
http://www.microsoft.com/downloads/details.aspx?familyid=e719ecf7-9f46-4312-af89-6ad8702e4e6e

Nota: Caso apareça a mensagem de erro da **Figura 6**, não se preocupe, você apenas está usando a versão 1.1 do *framework* para a aplicação ASP.NET. Para contornar, acesse o diretório virtual no IIS e na aba ASP.NET altere *ASP.NET version* para 2.0.50727.

A primeira funcionalidade que usaremos é saber que linhas foram "selecionadas" com o *CheckBox*. Adicione um botão e um *TextBox* na página. No *Click* do botão adicione o código da **Listagem 1**.

O código está comentado para um fácil entendimento. O que fizemos foi criar uma variável do tipo *StringBuilder* e adicionar um texto inicial, percorrer todas as linhas do *GridView* (utilizando o *for in*) e procurar o *CheckBox* que adicionamos via *template*, com o *FindControl*.

Esse método retorna um *Control*, então fizemos um *cast* para um *CheckBox*. Vale salientar que o parâmetro do *FindControl* é o nome do controle e o mesmo deve ser digitado corretamente, claro.

Se o controle não for nulo (foi encontrado) e estiver marcado (*Checked*), então adicionamos na *sb* os valores das colunas 1 e 2.

Nota: As colunas do *GridView* começam com 0, sendo que a do *TemplateField* é contada também.

Rode a aplicação, marque alguns *Check Boxs* e clique no botão. Como mostra a **Figura 7**, o *TextBox* exibe as linhas "selecionadas".

Como estamos usando AJAX, não temos *refresh* total da página. A funcionalidade de adicionar controles no *GridView* é semelhante para outros tipos de controles. Você pode tomar por base essas dicas para adicionar um *DropDownList*, por exemplo.

Master/Detail

Crie uma nova página e adicione dois *GridViews*. Para o primeiro, crie uma conexão com a tabela *Orders* do *Northwind*. Para o segundo *GridView* acesse os dados da tabela *Order Details*. Use o exemplo anterior para conectar ao banco e nas referidas tabelas.

Listagem 1. Obtendo a linha com o CheckBox marcado

```
uses System.Text;
...
var
  sb: StringBuilder;
  row: GridViewRow;
  ch: CheckBox;
begin
  { Cria um StringBuilder }
  sb := StringBuilder.Create;
  sb.Append('* Itens Selecionados *');
  { Percorre todas as linhas (rows) do GridView }

  for row in GridView1.Rows do
  begin
    { "procura" pelo CheckBox }
    ch :=  (row.FindControl('CheckBox1')) as CheckBox;
    { Se achou e está marcado }

    if ((ch <> nil) and (ch.Checked)) then
    begin
      { Adiciona na sb os campos do GridView }
      sb.AppendLine();
      sb.Append(row.Cells[1].Text + ' - ' + row.Cells[2].Text);
    end;

  end;
  { Repassa para o TextBox a sb }
  TextBox1.Text := '';
  TextBox1.Text := sb.ToString();
end;
```

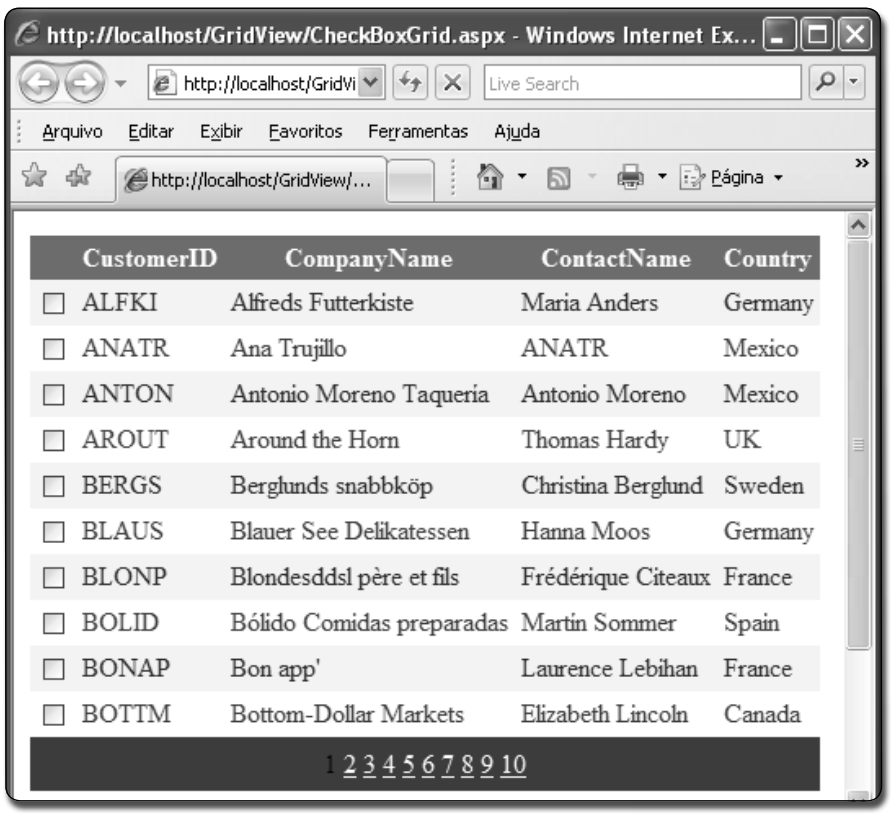


Figura 5. Controles CheckBoxes adicionados no GridView

Figura 6. Erro de configuração do framework

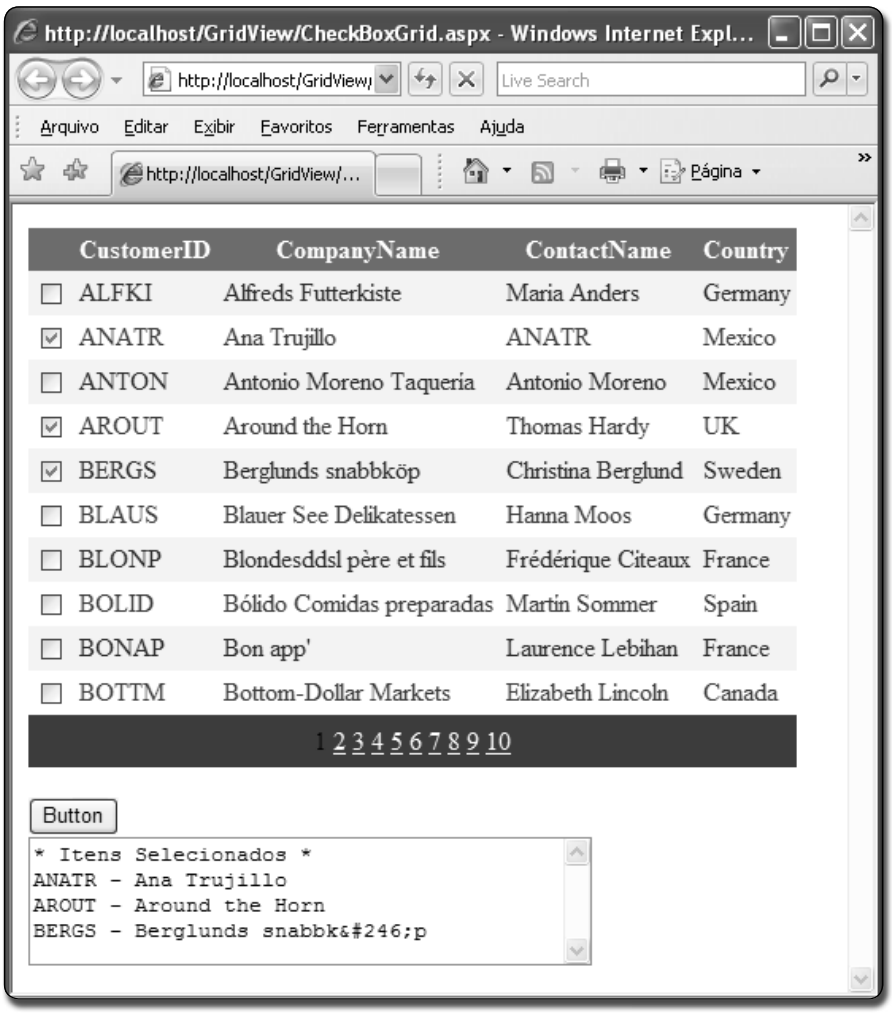


Figura 7. Obtendo os CheckBoxes marcados

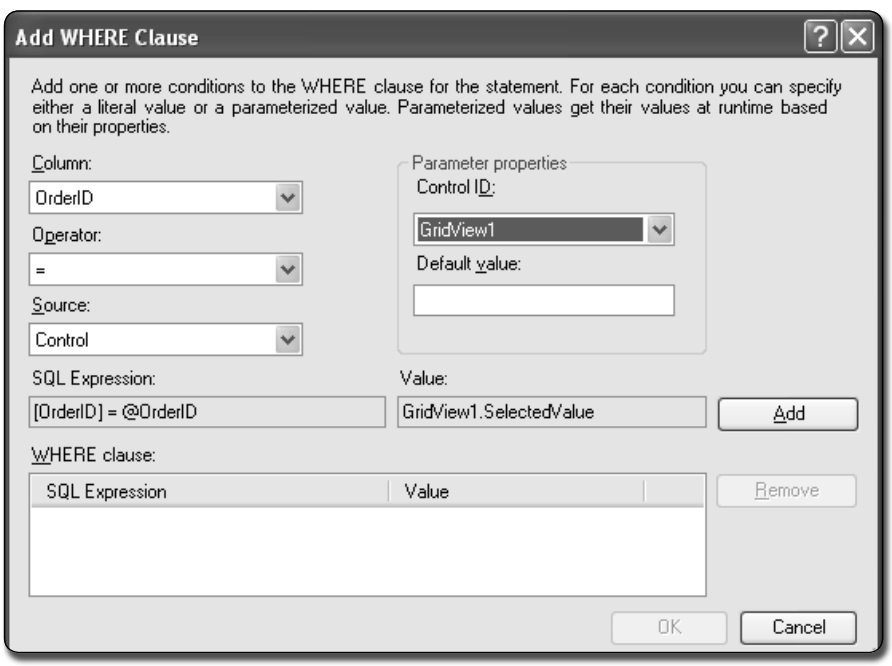


Figura 8. Configurando os parâmetros da consulta

Na tela do *wizard* onde você escolhe os campos, após escolher os mesmos da tabela *Order Details*, clique no botão *Where*. Na tela que abrir, em *Column* escolha *OrderID*, em *Source* selecione *Control* e em *ControlID* escolha *GridView1* (**Figura 8**).

Clique em *Add* para adicionar o parâmetro. Note que o valor do parâmetro será a propriedade *SelectedValue* do *GridView1*. Finalize o *wizard* e no *GridView1* acesse a *Smart Tasks*, marcando as opções *Enable Selection* e *Enable Paging*.

Configure as formatações dos controles para que fiquem diferentes um do outro. Rode a aplicação, selecione uma linha do *GridView1* e veja que o *GridView2* será preenchido com os dados referente aos itens da tabela pai (*Order*) (**Figura 9**).

Neste exemplo, configurei a consulta do *GridView2* para que trouxesse o nome do produto e a grande notícia que não usamos uma linha de código para fazer o *master/detail*, graças ao ASP.NET 2.0.

Master/detail com Templates

Mas se eu quiser que os itens (no caso os *details*) sejam mostrados sem que o usuário clique na linha do *GridView1*? Para isso usaremos *templates* no *GridView master*.

Crie uma nova página e adicione um *GridView*, adicionando os dados da tabela *Orders*. Acesse o editor de colunas do controle e remova todos os campos, adicionando apenas um do tipo *TemplateField*. Acesse as configurações do *template* e adicione uma tabela com duas linhas e três colunas.

Na linha superior, adicione o rótulo dos campos (digite ou coloque *Labels*), e na linha inferior, adicione três *Labels*. Para as *Labels* vamos usar a técnica de *DataBind* para mostrar os valores dos campos.

Acesse a *Smart Tasks* do primeiro *Label* e acesse a opção *Edit DataBindings*. Na janela que abrir, configure o campo no item *Bound to* (**Figura 10**).

Faça o mesmo para os outros campos. Abaixo da tabela, adicione um *GridView* e configure para acessar os dados da tabela *Orders Details* e configure um parâmetro igual ao exemplo anterior, só que ao invés de escolher o *GridView1*,

você escolherá o *Label1* (referente ao campo *OrderID*).

Finalize o *wizard* e rode a aplicação. Veja na **Figura 11** que estamos visualizando os dados da *master* (*Orders*) e em seguida a *detail* (*Order Details*).

Trabalhando com imagens

Mostrar imagens no *GridView* é simples. Por exemplo, para mostrar imagens nos comandos de seleção, exclusão, atualização e edição, basta configurar propriedades no editor de colunas, para o campo do tipo *CommandField*.

Indique a imagem que deseja na propriedade *XXImageURL*, onde o XX indica o tipo de comando (*Cancel*, *Delete*, *Insert*, *Update*). Por último, altere *ButtonType* para *Image*. Veja na **Figura 12** um exemplo dessas imagens. Também podemos traduzir os comandos na propriedade *XXText*.

Mas e se eu tenho imagens salvas na pasta da aplicação e na tabela do banco tenho apenas o nome ou caminho da mesma, como mostrar?

Vamos imaginar que tenhamos uma pasta no diretório da aplicação, chamada *imagens* e dentro dela os arquivos cadastrados no banco. Ao conectar a tabela, trazemos todos os campos da mesma.

No campo que guarda o nome da imagem, remova o mesmo e adicione um *ImageField*. Altere a propriedade *DataImageFieldURL* para o nome do campo da imagem. Rode a aplicação e veja as imagens sendo mostradas no *GridView* (**Figura 13**).

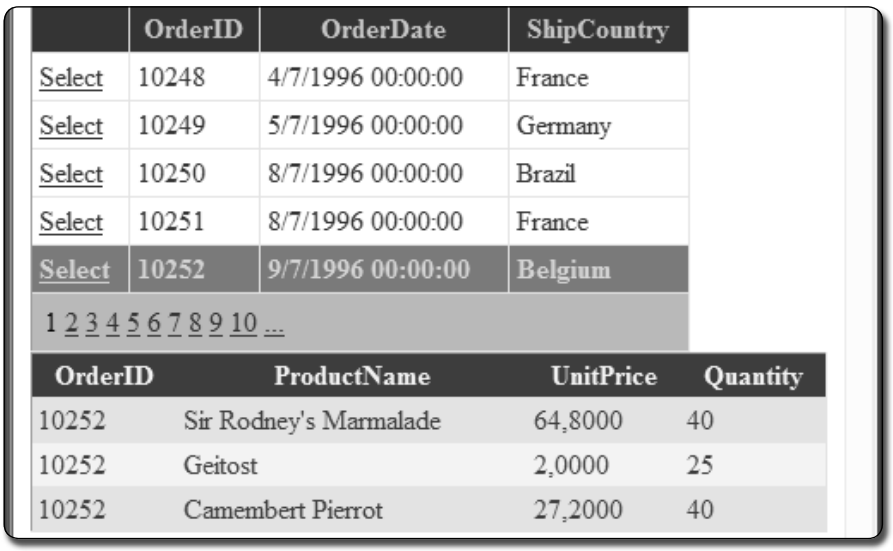
> **Nota:** No campo da tabela, você deve salvar o caminho correto da imagem, por exemplo: *imagens/nomedaimagem.jpg*.

Validações

Podemos usar os controles de validação normalmente na edição/inserção de valores no *GridView*. Novamente, faremos uso de *Templates*. Conecte a uma tabela do banco e escolha uma formatação para o *GridView*.

Transforme os campos que deseja colocar as validações, como *templates*. Acesse o editor de *templates* e escolha o item *EditItemTemplate* na opção *Display* (**Figura 14**).

Será mostrado o *TextBox* quando o usuário escolher a opção de inserção ou edição



	OrderID	OrderDate	ShipCountry
Select	10248	4/7/1996 00:00:00	France
Select	10249	5/7/1996 00:00:00	Germany
Select	10250	8/7/1996 00:00:00	Brazil
Select	10251	8/7/1996 00:00:00	France
Select	10252	9/7/1996 00:00:00	Belgium

1 2 3 4 5 6 7 8 9 10 ...

OrderID	ProductName	UnitPrice	Quantity
10252	Sir Rodney's Marmalade	64,8000	40
10252	Geitost	2,0000	25
10252	Camembert Pierrot	27,2000	40

Figura 9. Master/Detail com dois GridViews

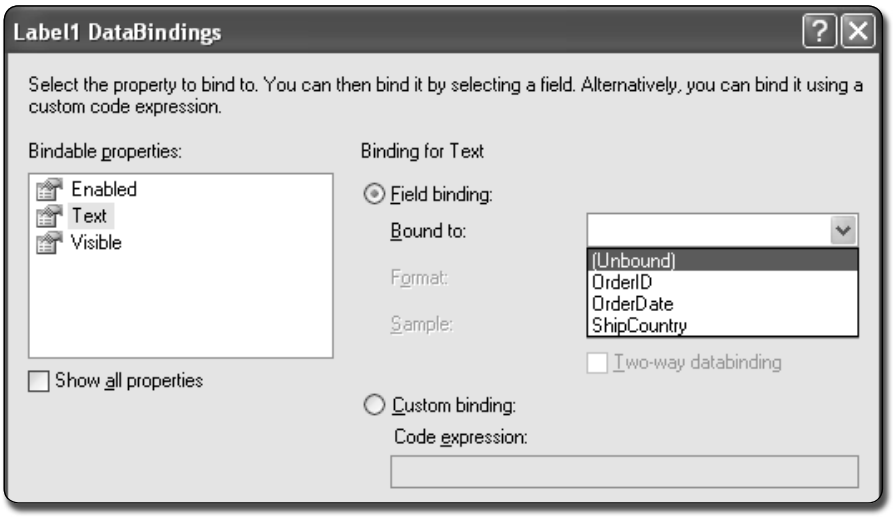


Figura 10. DataBinding de controles no TemplateField

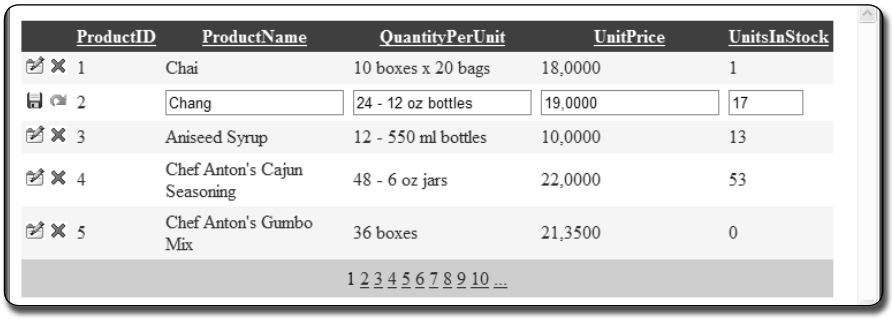
Master/Detail com Templates no GridView

OrderID	Order Date	Ship Country
10248	4/7/1996	France

OrderID	ProductName	UnitPrice	Quantity
10248	Queso Cabrales	14,0000	12
10248	Singaporean Hokkien Fried Mee	9,8000	10
10248	Mozzarella di Giovanni	34,8000	5

OrderID	Order Date	Ship Country
10249	5/7/1996	Germany

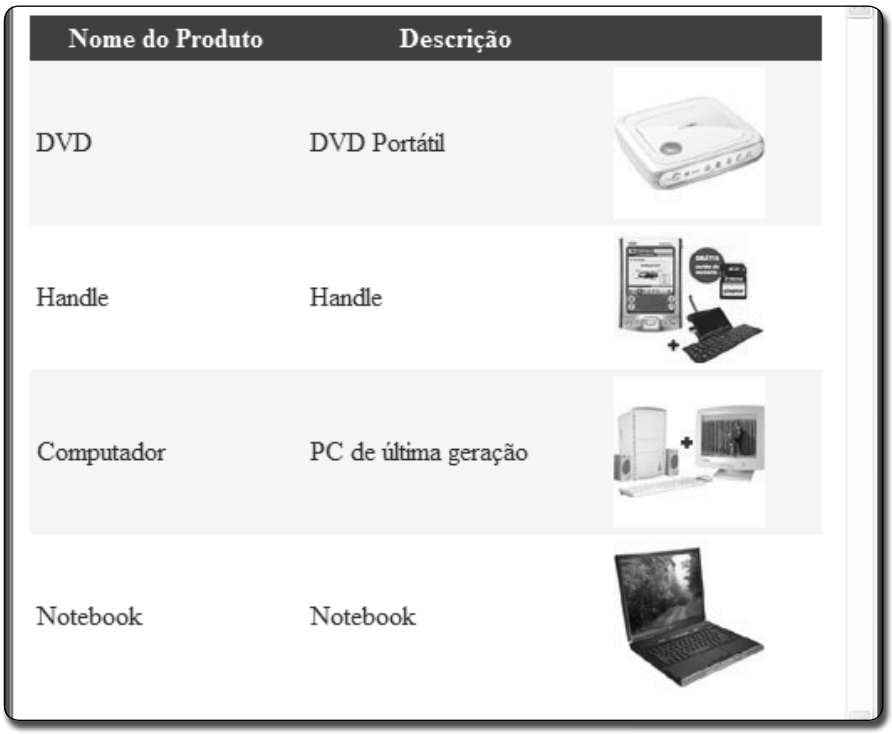
Figura 11. Master/detail usando templates



	ProductID	ProductName	QuantityPerUnit	UnitPrice	UnitsInStock
	1	Chai	10 boxes x 20 bags	18,0000	1
	2	Chang	24 - 12 oz bottles	19,0000	17
	3	Aniseed Syrup	12 - 550 ml bottles	10,0000	13
	4	Chef Anton's Cajun Seasoning	48 - 6 oz jars	22,0000	53
	5	Chef Anton's Gumbo Mix	36 boxes	21,3500	0

1 2 3 4 5 6 7 8 9 10 ...

Figura 12. Colocando imagens nos comandos do GridView



Nome do Produto	Descrição	
DVD	DVD Portátil	
Handle	Handle	
Computador	PC de última geração	
Notebook	Notebook	

Figura 13. Mostrando imagens da aplicação no GridView

da linha. Adicione um *RequiredFieldValidator* da aba *Web Validator* e aponte a propriedade *ControlToValidate* para o *TextBox*.

Digite a mensagem de erro em *ErrorMessage*. Feche o editor e rode a aplicação, clique em *Edit*, remova os valores e tente salvar (**Figura 15**).

Somatório no rodapé

Se o *GridView* possui um campo com valores numéricos ou monetários, certamente você gostaria de mostrar um total no rodapé da página. Primeiramente, altere para *True* a propriedade *ShowFooter* do *GridView*. Após, acesse o evento *RowDataBound* e adicione o código da **Listagem 2**.

Incrementamos uma variável apenas quando estivermos na linha do *GridView* (*DataRow*), colocamos o valor da mesma em uma variável de sessão. Senão, verificamos se estamos no rodapé do *GridView* e mostramos o valor. Assim, colocamos na primeira coluna, um texto e a quantidade (**Figura 16**). Se quiséssemos somar um valor do campo, bastaria "pegar" o valor usando: *e.Row.Cells[0].Text*, onde o 0 indica o índice da coluna.

Agrupamento

Para fazer um agrupamento, usaremos a mesma técnica já bastante mostrada neste artigo, *templates*. Será muito semelhante ao exemplo de *master/detail* usando *templates*. Para este exemplo, conectaremos o *GridView* à tabela *Categories* do *Northwind*.

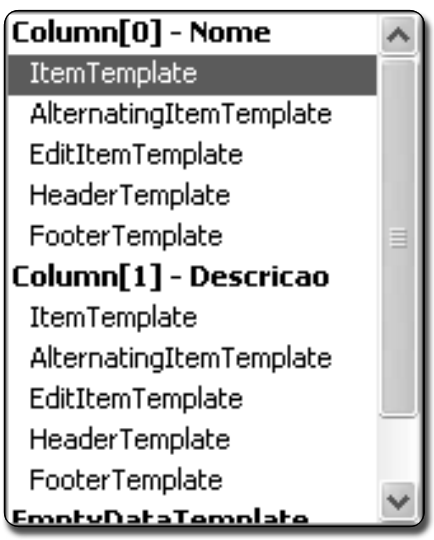
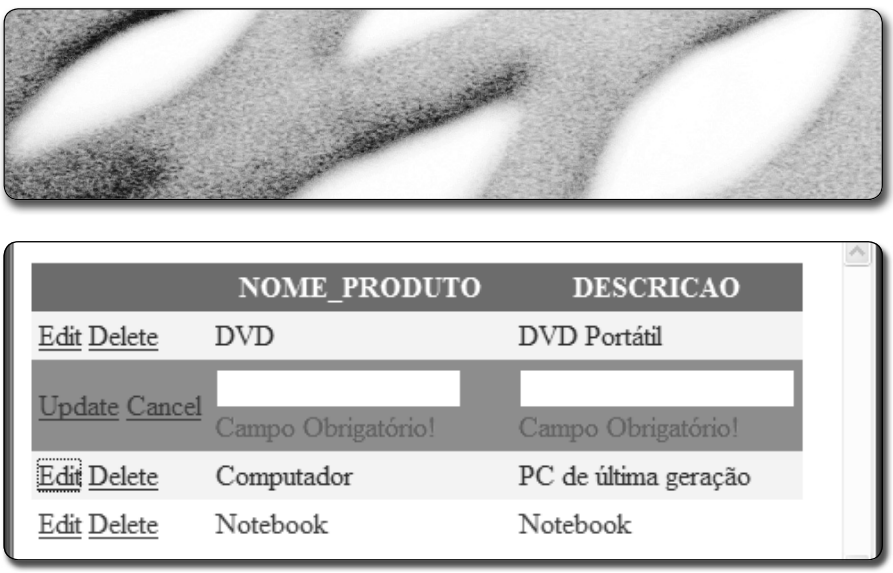


Figura 14. Opções do Template



	NOME_PRODUTO	DESCRICAO
Edit Delete	DVD	DVD Portátil
Update Cancel	Campo Obrigatório!	Campo Obrigatório!
Edit Delete	Computador	PC de última geração
Edit Delete	Notebook	Notebook

Figura 15. Validando os campos no GridView

Deixe apenas um campo e transforme-o em *TemplateField* no editor de colunas. Adicione um outro *GridView* no editor de *templates* e configure para a tabela *Products*, mas que mostre apenas os produtos de uma categoria específica (lembre-se do exemplo com parâmetros). Nesse caso, será um *Label* que você pode adicionar no editor de *templates*.

Depois basta rodar a aplicação e visualizar o agrupamento (**Figura 17**). Qualquer dúvida consulte o projeto que está para *download* (neste exemplo, removi o cabeçalho das colunas do *GridView* dos produtos).

Conclusão

Vimos neste artigo algumas dicas para a utilização do *GridView*, um *Grid* que será muito utilizado em aplicações ASP.NET com o RAD 2007. Para quem quiser avançar, que tal colocar todas essas dicas em prática, estendendo o controle padrão para um *GridView* turbinado. Fica aí a dica! Um grande abraço a todos e até a próxima! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

NOME_PRODUTO	DESCRICAO
DVD	DVD Portátil
Handle	Handle
Computador	PC de última geração
Notebook	Notebook
Total de itens: 4	

Figura 16. Mostrando o somatório no GridView

Listagem 2. Colocando um somatório no rodapé

```
var
  i: integer;
begin
  { se for a linha dos itens do GridView, faz a soma }
  if e.Row.RowType = DataControlRowType.DataRow then
  begin
    i := Convert.ToInt32(Session['i']) + 1;
    Session['i'] := &Object(i);
  end
  { se for o rodapé, mostra o resultado }
  else if e.Row.RowType =
    DataControlRowType.Footer then
    e.Row.Cells[0].Text := 'Total de itens: ' +
      Session['i'].ToString;
end;
```

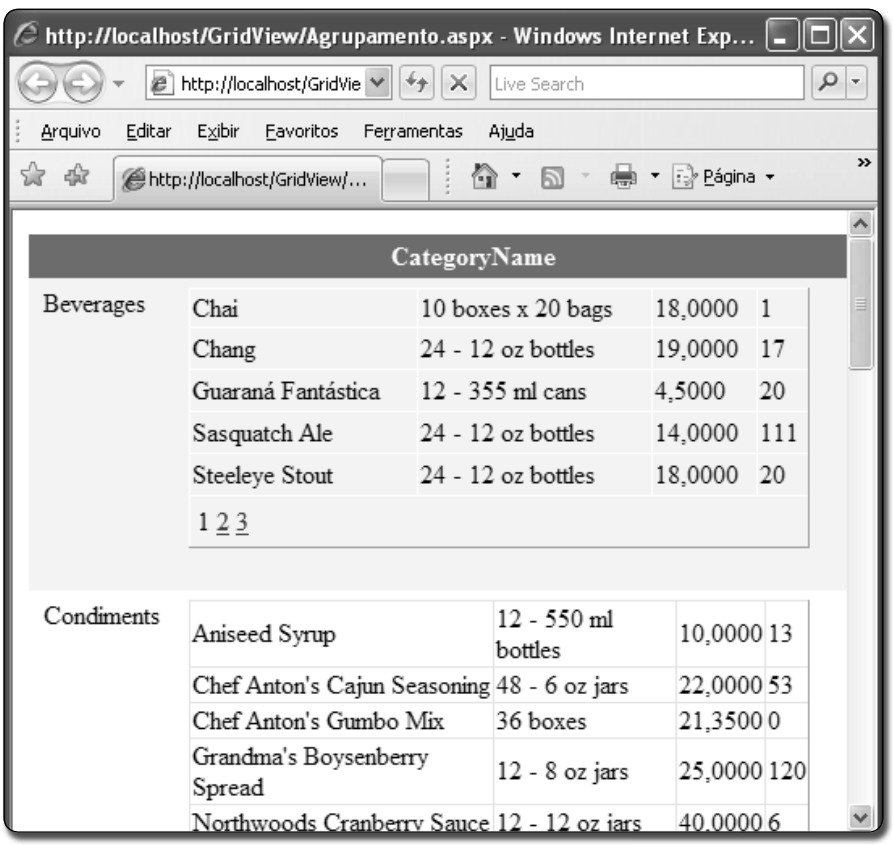


Figura 17. Agrupamento com dois GridViews

Nota do DevMan

As colunas do *GridView* possuem algumas propriedades para que possa ser personalizado. Vejamos algumas:

Tipos de botões: Selecione o controle *GridView* e clique no item *Smart Tasks>Edit Columns*. Na parte esquerda da tela localize o *CommandField* no grupo *Selected fields*. Observe que na parte direita, em *CommandField properties*, temos uma série de propriedades que podem ser modificadas. Uma delas, bastante interessante, é o *ButtonType*. Essa propriedade é responsável por exibir links para Editar, Alterar, Excluir, etc. Podemos escolher em as opções *Button*, *Link* e *Image*. Nesse ponto podemos utilizar um botão ao invés de uma imagem.

CancelText, EditText, InsertText, DeleteText, NewText e SelectText: Essas propriedades podem ser alteradas para traduzir os botões/links de comando. Por exemplo: ao invés de aparecer o texto Cancel em inglês, podemos traduzir para Cancelar.

ShowCancelButton, ShowDeleteButton, ShowEditButton e ShowInsertButton: Como os próprios nomes sugerem, podemos utilizar essas propriedades para mostrar/ocultar os botões *Cancel, Delete, Edit e Insert*.

Enfim, existem outras propriedades que podem ser usadas para personalizar a exibição do *GridView*. Modifique-as e execute a aplicação para ver o resultado.

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Controle on-line de vídeo-locadora

Veja como criar um sistema on-line de controle para uma vídeo-locadora - Parte 5

Resumo DevMan

Na última parte de nosso mini-curso veremos como bloquear novas reservas para clientes devedores. Esse recurso é bastante útil no dia-a-dia e evita que o cliente alugue muitos filmes sem antes eliminar todos os seus débitos.

Outro recurso interessante que veremos será incluído na área administrativa que possibilitará o operador da locadora bloquear, ou não, o cliente inadimplente ou com filmes em atraso.

Nesse artigo veremos

- Criação das páginas para liberação de locação;
- Bloqueio de locações para usuários devedores;
- Bloqueio de novas reservas para usuários devedores.

Qual a finalidade

- Com o bloqueio de reservas e multas para usuários devedores, evitamos que o mesmo faça novas reservas sem antes liquidar seus débitos e atrasos nas devoluções.

Quais situações utilizam esses recursos?

- A principal situação está no fato de evitar mais problemas com um determinado usuário que costuma atrasar a entrega do filme e não pagar as devidas multas.

Finalizando, neste artigo, o desenvolvimento da nossa aplicação ASP.NET com *Delphi for .NET*, sobre um sistema on-line de vídeo-locadora, veremos uma funcionalidade de grande valia para que você possa implementar em seu sistema, o controle de atrasos e multas. Criaremos uma nova tabela na base de dados onde armazenaremos todos os registros dos valores de multas aplicados a clientes decorrentes de atrasos na devolução de suas locações.

Para o processo de reservas de locações pelo cliente, faremos uma verificação no momento da reserva, onde apenas usuários sem registros de atrasos ou multas estarão aptos a novas reservas. Para o controle da locadora, nas páginas onde os administradores possuem acesso, criaremos também uma verificação no momento em que um usuário é selecionado, bem como a regra de criação de multa em devoluções de locações

Maikel Marcelo Scheid
maikelscheid@gmail.com
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

atrasadas, e posterior criação de páginas para pagamento das mesmas. Veja que as implementações não terão grandes dificuldades ao serem desenvolvidas. Mãos na massa!

Adicionando a tabela de multas ao banco de dados

Utilizando o gerenciador de banco de dados *IBExpert,* ative o registro da base de dados da locadora on-line e abra o *SQL Editor* ("F12") onde iremos digitar os comandos *SQL* para criação da tabelas de "MULTAS", *Primary Keys* e *Foreign Keys*, além também da criação de um mecanismo de auto-incremento para a chave primária da tabela.

Digite no editor *SQL* a instrução da **Listagem 1** e logo após execute-a ("F9"). Para compilar e gravar a ação utilize o botão *Commit Transaction (Ctrl+Shift+C),* observando no *DataBase Explorer* que a nova tabela acabou de ser adicionada.

Ao observar o código, veja que nas primeiras linhas estamos criando as tabelas e os respectivos campos, definindo a cada um dos mesmos um tipo e tamanho se necessário. Na seqüência, a criação da chave-primária da tabela, à qual também atribuímos um *Generator* e uma *Trigger* que irá definir um valor automático para o campo, caso o mesmo não seja informado no momento de uma nova inclusão. Na segunda parte da instrução, criamos uma chave estrangeira relacionada à tabela *LOCACAO* por onde conseguiremos obter o registro causador da multa, nome do vídeo que está atrasado e cliente da locação. Realizadas as mudanças na base de dados, passaremos à implementação das rotinas de programação na aplicação.

Criando uma unit para controle de multas e atrasos

No *Delphi 2006*, abra o projeto que estamos desenvolvendo e crie uma nova *Unit* através do *menu File>New>Other>Delphi for .NET Projects>New Files>Unit*. Nesta *Unit* criaremos duas funções e um procedimento, onde, as funções farão verificações de registros de locações em atraso de um usuário e dos registros de multas sem pagamento. O procedimento será utilizado para adicionar novos registros de multas no momento em que uma devolução for feita com atraso.

Criada a *Unit* salve-a com o nome *uControleMultas.pas* e após isso adicione os códigos de acordo com a **Listagem 2** onde todas as linhas codificadas encontram-se comentadas de acordo com a ação que executa. Adicionaremos também nesta *Unit* uma constante contendo o *Path* de conexão com a base de dados. Durante a codificação das *functions* e *procedures* note que também estamos declarando todas as variáveis de comunicação com a base de dados, que serão criadas e configuradas em tempo de execução.

A partir destas funções e procedimento teremos todo o controle relacionado aos atrasos e multas de usuários, bastando que agora sejam codificadas as páginas para que passem a utilizar as verificações criadas nesta *Unit*. Nosso primeiro passo será realizar as verificações na página de acesso do cliente, onde ele próprio tem a opção de realizar uma nova reserva, opção que passará a ser bloqueada caso alguma irregularidade seja encontrada. Antes disso vejamos o que estamos fazendo em cada procedimento.

A função *UsuarioPossuiLocacaoAtrasada* será utilizada para verificar se o usuário possui uma locação atrasada conforme o próprio nome diz. Em primeiro lugar estamos criando um componente de conexão, (*Conn*) atribuindo a ele a *string* de conexão, e logo em seguida, conectamos a base de dados. Na seqüência atribuímos a propriedade *SelectCommand* do *DataAdapter* o comando SQL para seleção dos vídeos. Note que estamos fazendo um *Inner Join* com a tabela *Locacao,* assim conseguimos saber se este usuário possui alguma locação em aberto. Adicionamos um parâmetro ao *DataAdapter* e atribuímos a ele o *ID* do usuário para que possamos abrir a pesquisa.

Um detalhe que deve ser observado é nossa cláusula *Where*. Nela estamos filtrando todos os filmes que têm *Status* igual a *N*, data de devolução menor que a data atual além, é claro, de filtrar por usuário. Por fim imprimimos o resultado na tela usando um laço *while*. Perceba que usamos comando HTML ("
" e "
") no laço para criar uma lista.

Outra função codificada foi a *UsuarioPossuiMultaNaoPaga,* que é bastante semelhante a que vimos anteriormente, tendo como diferença apenas a cláusula *Where.* Neste caso estamos filtrando o usuário e todas as locações que possuem o campo *Pago* igual a *N.*

Por último vejamos a função *GravaMultaUsuario,* responsável por calcular e gravar a multa por atraso. Repetimos a criação dos componentes de conexão e pesquisa e conectamos ao banco. Dessa vez usamos a propriedade *InsertCommand*, já que precisamos gravar os dados de multa da base de dados. Nesse momento estamos usando a *Trigger Multas* para fazer a inserção da respectiva multa na tabela de mesmo nome. Como parâmetro, usamos o próprio *ID* da locação.

Listagem 1. Adicionando tabela de multas à base de dados

```
CREATE TABLE MULTAS(
  COD_MULTA INTEGER NOT NULL,
  COD_LOCACAO INTEGER,
  PAGO CHAR(1) DEFAULT 'N',
  DATA_PGTO DATE DEFAULT 'TODAY',
  VALOR DECIMAL(12,2) DEFAULT 0);
ALTER TABLE MULTAS
ADD CONSTRAINT PK_MULTAS
PRIMARY KEY (COD_MULTA);
CREATE GENERATOR GEN_MULTAS_ID;
SET TERM ^;
CREATE TRIGGER MULTAS_BI FOR MULTAS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD_MULTA IS NULL) THEN
    NEW.COD_MULTA = GEN_ID(GEN_MULTAS_ID,1);
END
^
SET TERM ; ^
ALTER TABLE MULTAS
ADD CONSTRAINT FK_MULTAS
FOREIGN KEY (COD_LOCACAO)
REFERENCES LOCACAO(COD_LOCACAO);
```

Verificando existência de atrasos e multas na página do cliente

Abra a página *user_reservas.aspx* criada na parte dois do nosso curso e clique duas vezes no botão *btnReservar* para que possamos ter acesso ao evento *Click* do botão. Faremos uma pequena modificação. Realize a declaração de duas variáveis do tipo *string* ("Atrasos" e "Multas"). O código já existente deverá ser incluído em um bloco *begin..end*. Esse bloco será o resultado de uma condição. Ele só será executado se não houver atrasos ou multas. Adicione a referência para a *Unit* dos controles (*Alt+F11*) e logo nas primeiras linhas da codificação adicione o seguinte código:

```
Atrasos := UsuarioPossuiLocacaoAtrasada(
  Session['CODIGO'].ToString);
Multas  := UsuarioPossuiMultaNaoPaga(
  Session['CODIGO'].ToString);
if (Atrasos = '') and (Multas = '') then
begin
  //Código anteriormente digitado
end;
```

O código adicionado fará a verificação relacionada ao usuário logado, se o mesmo possui alguma irregularidade de locação em atraso ou alguma multa em pendência. Ao final do código já existente, faremos uma implementação que em caso afirmativo de irregularidade, as mesmas serão listadas em um

Listagem 2. Unit para controle de multas e atrasos

```
unit uControleMultas;
interface
uses
  FirebirdSQL.Data.Firebird, Borland.Data.Common;
{ Função que irá retornar a lista de locações onde a data para
  devolução é menor que a data de hoje -> atrasados }
function UsuarioPossuiLocacaoAtrasada(idUser:  string): string;
{ Irá retornar a lista dos registros de multas junto com as
  referentes locações já devolvidas, mas não pagas. }
function UsuarioPossuiMultaNaoPaga(idUser:  string): string;
{ Cria novo registro de multa para usuário. }
procedure GravaMultaUsuario(idLocacao:  string; valor: decimal);

implementation

const
  strConexao = 'User=SYSDBA;Password=masterkey;
  Database=C:\Inetpub\wwwroot\VideoLocadora\LOCADORA.FDB;';

function UsuarioPossuiLocacaoAtrasada(idUser: string): string;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  DataReader : FbDataReader;
  Conn: FbConnection;
  prUser : FbParameter;
begin
  Result := '';
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da  string de conexão e abertura do BD }
  Conn.Connection string := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT VIDEOS.TITULO, ' +
    'LOCACAO.PREV_DEVOLUCAO FROM LOCACAO ' +
    '   INNER JOIN VIDEOS ON (LOCACAO.VIDEO =' +
    'VIDEOS.COD_VIDEO) ' +
    'WHERE ((LOCACAO.STATUS = ''N'') AND ' +
    '(LOCACAO.PREV_DEVOLUCAO < ''TODAY'') '+
    'AND (LOCACAO.CLIENTE = ?))';
  { Criação dos parâmetros }
  prUser := FbParameter.Create;
  { Atribuição de valores aos parâmetros }
  DataAdapter.SelectCommand.Parameters.Add(prUser);
  DataAdapter.SelectCommand.Parameters[0].Value := idUser;
  { Executando o comando }
  DataReader := DataAdapter.SelectCommand.ExecuteReader;
  { Enquanto houver registros no resultado faz um laço criando a lista }
  while DataReader.Read do
  begin
    Result := Result + '<li>' + DataReader[0].ToString
    + ' (' + DataReader.GetDateTime(1).ToString('dd/MM/yyyy')
    + ')</li><br>';
  end;
end;

function UsuarioPossuiMultaNaoPaga(idUser: string): string;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  DataReader : FbDataReader;
  Conn: FbConnection;
  prUser : FbParameter;
begin
  Result := '';
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da  string de conexão e abertura do BD}
  Conn.Connection string := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT LOCACAO.DATA_FINAL, ' +
    'VIDEOS.TITULO, MULTAS.VALOR FROM MULTAS ' +
    '   INNER JOIN LOCACAO ON (MULTAS.COD_LOCACAO =' +
    'LOCACAO.COD_LOCACAO) ' +
    '   INNER JOIN VIDEOS ON (LOCACAO.VIDEO = VIDEOS.COD_VIDEO) ' +
    'WHERE ((MULTAS.PAGO = ''N'') AND (LOCACAO.CLIENTE = ?))';
  { Criação dos parâmetros }
  prUser := FbParameter.Create;
  { Atribuição de valores aos parâmetros }
  DataAdapter.SelectCommand.Parameters.Add(prUser);
  DataAdapter.SelectCommand.Parameters[0].value := idUser;
  { Executando o comando }
  DataReader := DataAdapter.SelectCommand.ExecuteReader;
  { Enquanto houver registros no resultado faz um laço criando a lista }
  while DataReader.Read do
  begin
    Result := Result + '<li>' + DataReader.GetDateTime(0).ToString
      ('dd/MM/yyyy') + ' - ' + DataReader[1].ToString + ' [' +
      DataReader.GetDecimal(2).ToString('#0.00') + ']</li><br>';
  end;
end;

procedure GravaMultaUsuario(idLocacao:  string;  Valor: Decimal);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  DataReader : FbDataReader;
  Conn: FbConnection;
  prLocacao : FbParameter;
  prValor : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.Connection string := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.InsertCommand := Comand;
  DataAdapter.InsertCommand.Connection := Conn;
  DataAdapter.InsertCommand.CommandText :=
    'INSERT INTO MULTAS (COD_LOCACAO, VALOR) VALUES (?, ?)';
  { Criação dos parâmetros }
  prLocacao := FbParameter.Create;
  { Atribuição de valores aos parâmetros }
  DataAdapter.InsertCommand.Parameters.Add(prLocacao);
  DataAdapter.InsertCommand.Parameters[0].value := idLocacao;
  { Criação dos parâmetros }
  prValor := FbParameter.Create;
  { Atribuição de valores aos parâmetros }
  DataAdapter.InsertCommand.Parameters.Add(prValor);
  DataAdapter.InsertCommand.Parameters[1].
    value := valor.To string.Replace(',','.');
  { Executando o comando }
  DataAdapter.InsertCommand.ExecuteNonQuery;
end;
end.
```

Label(*lblAtrasos*), que deverá ser adicionado logo abaixo ao botão *btnReservar*. O código a ser adicionado ao final está descrito a seguir:

```
end
else
begin
  lblAtrasos.Text :=
    'Reserva não pode ser concluída.<br>'+
    'ATRASOS:<br>'+Atrasos+'<br>'+
    ' MULTAS:<br>'+Multas;
end;
```

Caso o usuário possua atrasos e/ou multas, mensagens serão impressas na página logo abaixo da listagem de locações. A listagem completa poderá ser baixada do site da DevMedia no link desse artigo (**Figura 1**).

Verificando existência de atrasos e multas no modo administrador

Abrindo a página *a_locacao.aspx* faremos a implementação de duas verificações, onde uma delas será relacionada ao usuário que está a locar um vídeo, controle muito semelhante ao aplicado na página do cliente. Outra verificação será no momento da devolução de obras, que por sua vez estando atrasada irá gerar automaticamente um registro de multa relacionado.

Para a verificação da situação do cliente, que somente os administradores poderão ter acesso, não bloquearemos um empréstimo, mas sim vamos alertar ao operador do sistema que alguma irregularidade existe com o usuário relacionado. Esse operador terá plenos poderes para pausar ou prosseguir o empréstimo. No componente *ddlClientes* onde a lista de clientes é preenchida, selecione a propriedade *AutoPostBack* alterando-a para *True*, mudança que fará com que toda vez que um cliente for selecionando, um código de verificação seja disparado automaticamente.

No evento *SelectedIndexChanged* adicione o código da **Listagem 3**, sendo este o responsável por exibir a situação cadastral do cliente em um *Label*(*lblSituacao*) que deverá ser adicionado logo abaixo do *ddlClientes* da estrutura da página.

A codificação da verificação é bastante simples, sendo que duas variáveis são declaradas e recebem o valor das funções que criamos para verificação de atrasos e multas. Para cada uma das variáveis declaradas, caso apresente algum valor, serão exibidas como irregularidade para o referido cliente (**Figura 2**), ou então, como cliente com situação normalizada.

Outra verificação que faremos nesta página está relacionada às devoluções de vídeos, onde no momento em que o administrador clicar em "devolver" uma comparação entre a data atual e a data de devolução prevista será realizada. Se a data atual superar a data prevista, indica que a devolução está atrasada e que um registro de multa deverá ser

Figura 1. Locações atrasadas e registro de multas

Figura 2. Verificação da situação cadastral de clientes

Listagem 3. Verificando situação cadastral de cliente

```
procedure TWebForm1.ddlClientes_SelectedIndexChanged(
  sender: System.Object; e: System.EventArgs);
var
  Atrasos :  string;
  Multas  :  string;
begin
  { Declare a Unit de controle de multas uControleMultas.pas }
  Atrasos := UsuarioPossuiLocacaoAtrasada(ddlClientes.SelectedValue);
  Multas  := UsuarioPossuiMultaNaoPaga(
    ddlClientes.SelectedValue);
  if (Atrasos = '') and (Multas = '') then
  begin
    { Situação do cliente OK }
    lblSituacao.Text := 'Cliente sem irregularidades!'
  end
  else
  begin
    { Exibe na tela as irregularidades do cliente }
    lblSituacao.Text := 'Irregularidades do Cliente:<br>'+
    'ATRASOS:<br>' + Atrasos + '<br>' + ' MULTAS:<br>' + Multas;
  end;
end;
```

criado. No evento *ItemCommand* do *gridLocacao* adicione o código da **Listagem 4**, deixando parte do código já existente, onde a verificação dos dias de atraso e o cálculo do valor da multa são realizados, alertando os mesmos ao administrador do sistema. Neste caso, um valor fixo de *R$ 0,50* foi estabelecido para cada dia de atraso, mas que você poderá customizar e deixar como campo de preenchimento de forma dinâmica pelo sistema.

Quitando registro de multas

A quitação dos registros de multas de usuário será realizada nas páginas de acesso exclusivo de administradores da locadora, onde precisaremos criar uma nova página *a_multas.aspx* e adicionar um botão (*btnMultas*) em *principal.aspx* de acesso do administrador que, ao ser clicado, irá redirecionar o mesmo à página das manutenções.

Com a página de manutenções de multas criada, da mesma maneira que criamos todas as outras páginas anteriores, crie uma tabela de três linhas e uma coluna, onde na primeira linha da tabela arraste o *User Control cabecalho.ascx*. Na próxima linha adicione um componente *DataGrid*(*gridMultas*) alterando sua propriedade *width* para *100%* da largura e com o botão direito do mouse sobre o mesmo. Selecione *Property Builder* e selecione a guia *Columns* onde deverá adicionar um *Button Column>Select* alterando os *Header Text, Text e Command Name* para "Pagar" que será a coluna utilizada para ativar o pagamento ao clique na mesma. Na última linha da tabela adicione um botão (*btnVoltar*) que no seu código redirecione o usuário de volta à página principal dos administradores.

No código da página, criaremos um procedimento que irá selecionar todos os registros de multas não pagas e permitir que sejam quitados. Declare no método *public* do código da página a *procedure* "*CarregarMultas*" e adicione o código da **Listagem 5,** e igualmente em todas as outras páginas, adicione a constante de conexão contendo o caminho físico da base de dados que será utilizada para a conexão. Declaramos também variáveis com os objetos que serão criados para a comunicação com a base de dados.

Na **Listagem 5** criamos o componente para acesso a dados e atribuímos a ele a *string* de conexão, assim como temos feito até o momento. Em seguida montamos uma instrução SQL para trazer os dados da tabela *Multas* com seus respectivos clientes. Note que estamos fazendo *Join* com as tabelas *Multas*, *Clientes*, *Locacao* e *Videos*.

Listagem 4. Verificando atraso e criando registro de multa

```
procedure TWebForm1.gridLocacao_ItemCommand(source: System.Object; e:
  System.Web.UI.WebControls.DataGridCommandEventArgs);
var
  valor : decimal;
  dias  : integer;
begin
  { Verifica se a data de hoje é maior que a previsão de devolução }
  {Para usar o Convert.ToDateTime declare o namespace Borland.Vcl.SysUtils }
  if Date.Now > Convert.ToDateTime(e.Item.Cells[3].Text.ToString) then
  begin
    { Calcula a quantidade de dias do atraso }
    dias := DaysBetween(Date.Now,Convert.ToDateTime(e.Item.Cells[3].Text.ToString));
    { Calcula valor da multa }
    Valor := 0.5 * dias;
    { Grava o registro de multa }
    GravaMultaUsuario(e.Item.Cells[0].Text.ToString, Valor);
    { Alerta na tela que registro de multa foi adicionado }
    RegisterStartupScript('multa','<script>javascript:alert(''Dias de atraso: ' +
      dias.To string + '\n' + 'Valor da multa: R$ '+valor.To string+''');
      </script>');
  end;
  { Faz a devolução do vídeo }
  DevolverVideo(e.Item.Cells[0].Text);
end;
```

Listagem 5. Listando todos os registros de multas

```
procedure TWebForm2.CarregarMultas;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
begin
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.Connection string := strConexao;
  Conn.Open;
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT MULTAS.COD_MULTA, LOCACAO.DATA_FINAL, ' +
    '  CLIENTES.NOME, VIDEOS.TITULO, MULTAS.VALOR ' +
    'FROM MULTAS '+
    '  INNER JOIN LOCACAO' +
    '  ON (MULTAS.COD_LOCACAO = LOCACAO.COD_LOCACAO)'+
    '  INNER JOIN CLIENTES' +
    '  ON (LOCACAO.CLIENTE = CLIENTES.COD_CLIENTE) ' +
    '  INNER JOIN VIDEOS' +
    '  ON (LOCACAO.VIDEO = VIDEOS.COD_VIDEO) ' +
    'WHERE ((MULTAS.PAGO = ''N''))';
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');
  try
    gridMultas.DataSource := Ds;
    gridMultas.DataBind;
  finally
    Conn.Close;
  end;
end;
```

Nota: Não esqueça de adicionar os *namespaces FirebirdSQL.Data.Firebird* e *Borland.Data.Common* para que tenha acesso às funções de conexão com o BD.

Como dito anteriormente, código faz a criação e configuração dos componentes de conexão e consulta ao banco de dados, trazendo resultados através de um comando SQL entre várias tabelas "amarradas" através de seus campos chave e chaves estrangeiras. Os resultados obtidos são preenchidos em um *DataSet* que é, por sua vez, listado na tela através do *Grid*. Para que o código seja executado, dê um clique duplo na área da página e adicione no *Page_Load* a seguinte linha que irá chamar o procedimento e preencher o *Grid*.

```
CarregarMultas;
```

O pagamento de multas será realizado pelo clique na coluna "Pagar"

adicionada no *Grid*, onde o código da **Listagem 6** deverá ser adicionado ao seu evento *ItemCommand*, o qual irá enviar ao banco de dados uma instrução *SQL* carregada com um parâmetro de código da multa para que o *Status* da mesma seja alterado para *S* ("Pago").

Após criar os componentes de acesso ao banco e execução de *scripts*, veja o comando *Update* que estamos usando para atualizar o banco de dados.

```
UPDATE MULTAS SET PAGO = ''S''
  WHERE (COD_MULTA = ?)
```

Nota do DevMan

Durante todo o mini-curso criamos, em tempo de execução, os componentes de conexão com o banco de dados assim como os componentes para seleção, inclusão ou alteração do BD. Incluímos também, em todas as páginas, uma constante com a string de conexão. Esses recursos são interessantes e agilizam o carregamento da página, porém há uma repetição enorme de código e a declaração fixa da constante de string de conexão em todas as páginas, como dito anteriormente. Em uma eventual troca de servidor de banco de dados nós teremos que, obrigatoriamente, modificar essa string de conexão página a página. Em um sistema pequeno, com poucas telas, isso torna-se bastante confortável. Contudo, em um sistema complexo com muitas telas e múltiplas conexões esse esquema pode se tornar um tormento, visto que podemos esquecer de alterar em algum lugar.

Pensando nisso tudo, podemos criar uma Unit onde podemos centralizar nossa string de conexão e até criar funções para conexão de desconexão com o BD. Desta forma temos total segurança na hora de fazer algum tipo de modificação na conexão.

Ainda com esse pensamento, podemos utilizar a Unit criada para centralizar nossas possíveis regras de negócios, tais como: verificação de atrasos, verificação de débito, checagem de usuário e senha no login, etc.

Enfim, há muito o que fazer para melhorar o desenvolvimento e agilizar determinados processos em toda a criação e manutenção do sistema.

E finalmente, uma outra opção, a mais utilizada, é armazenar a string de conexão no Web.Config. Exemplo:

```
Web.config:

<appSettings>
<add key="FbCon" value="
  User=SYSDBA;Password=masterkey;
  Database=C:\Inetpub\wwwroot\
  VideoLocadora\LOCADORA.FDB " />
</appSettings>

App:
var
  configurationAppSettings: System.
    Configuration.AppSettingsReader;
begin
  configurationAppSettings := System.
   Configuration.AppSettingsReader.Create;
  FbConnection1.ConnectionString :=
   (string(configurationAppSettings.
   GetValue('FbCon', TypeOf(string))));
```

Ao executar a página, vá até a área de *login* de administradores e acesse a seção de multas, visualizando a listagem das multas geradas pelo sistema (**Figura 3**) e confirme o pagamento das mesmas.

Conclusão

Finalizamos, neste controle de multas, a última parte do nosso artigo de vídeo-locadora, onde a maioria dos processos rotineiros relacionados à atividade estão sendo contemplados. Obviamente, muitas customizações ainda poderão ser implementadas, como relatórios de gerenciamento ou acompanhamento de usuários, manutenção de valores por atrasos, condições de pagamento, entre outros. Espero que tenham gostado da seqüência de artigos e assuntos apresentados, grande abraço e até o próximo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nota do Editor: Todo este mini-curso foi construído baseado no ASP.NET 1.1, suportado pelo *Delphi 8, 2005* e *2006*. Como sabemos, o RAD 2007 trouxe suporte ao ASP.NET 2.0, e muitas coisas foram introduzidas, outras mudaram. Os conceitos aqui apresentados valem então para a utilização do ASP.NET versão 1.1. Em vários outros artigos aqui da revista vamos abordar o ASP.NET 2.0 e todos os seus novos recursos.

Figura 3. Listagem de multas

Listagem 6. Pagamento de multas

```
procedure TWebForm2.gridMultas_ItemCommand(source: System.Object; e:
  System.Web.UI.WebControls.DataGridCommandEventArgs);

var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  prCod_Multa : FbParameter;

begin
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.Connection string := strConexao;
  Conn.Open;

  DataAdapter.UpdateCommand := Comand;
  DataAdapter.UpdateCommand.Connection := Conn;
  DataAdapter.UpdateCommand.CommandText := 'UPDATE MULTAS SET PAGO = ''S'' WHERE (COD_MULTA = ?)';

  prCod_Multa := FbParameter.Create;
  DataAdapter.UpdateCommand.Parameters.Add(prCod_Multa);
  DataAdapter.UpdateCommand.Parameters[0].Value := e.Item.Cells[1].Text;

  if DataAdapter.UpdateCommand.ExecuteNonQuery> 0 then
    CarregarMultas;
end;
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Sistema DataCar em OO

Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 2

Resumo DevMan

Nesta parte do curso aprenderemos um pouco sobre mapeamento objeto relacional (O/R), persistência de dados e também sobre o padrão DAO.

Nesse artigo veremos

- Geração das classes de nosso sistema DataCar;
- O que é Mapeamento Objeto-Relacional;
- Implementação de Relacionamentos.

Qual a finalidade

- Permitir a construção do núcleo do sistema.

Quais situações utilizam esses recursos?

- As classes criadas aqui farão parte do núcleo do sistema que será todo orientado a objetos. Aprenderemos que um sistema OO inicia-se pelas classes, diferentemente do que já estamos acostumados, ou seja, começar pelo banco de dados. Também veremos o padrão DAO.

Acredito que uma boa parte dos programadores de hoje em dia ainda desenvolvem suas aplicações de modo *procedural* ou *procedimental*. E isso é uma enorme barreira para quem deseja se aventurar e desenvolver sistemas totalmente orientados a objetos. Há um salto bastante enorme de conceito entre *Programação Procedural* e *Programação Orientada a Objetos*. O primeiro grande passo a se tomar é entender que um sistema OO nasce de classes e não do banco de dados como estamos acostumados a fazer em nosso dia-a-dia. Veremos em nosso sistema *DataCar* como romper essa barreira e entender completamente o ciclo de um *software* OO.

Na primeira parte do artigo vimos como criar e testar usando o *DUnit* nossa primeira classe do sistema. Nesta parte do artigo vamos abordar o seguinte problema: "como vamos salvar nossos objetos?". Solucionar essa dúvida implica entrar em dois assuntos, mapeamento objeto relacional e estratégia de persistência. Abordemos então o primeiro assunto.

Paulo Roberto Quicoli
pauloquicoli@gmail.com
é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente-servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP. Blog: http://pauloquicoli.spaces.live.com.

Mapeamento objeto-relacional e desencontro objeto-relacional

O mapeamento objeto-relacional é, de forma resumida, a tarefa de se dizer a um sistema onde determinada classe será salva em um banco de dados relacional, como uma herança pode ser representada, como identificar um objeto dentro do SGBD. Existe uma diferença entre classes e tabelas, a essa diferença dá-se o nome de (desencontro objeto-relacional ou impedância objeto-relacional).

Essencialmente a programação Orientada a Objetos é diferente da programação Orientada a Dados. Quando se programa pensando em objetos levamos em conta seu comportamento e seus dados, diferentemente em um desenvolvimento centrado a dados, levamos em conta apenas a melhor forma de se armazenar os dados, a lógica do negócio envolvido fica separada dos dados.

Mesmo tendo um banco de dados relacional podemos unir dados e comportamento em classes, isto através do mapeamento relacional.

A importância dos OIDs

Para minimizar o impacto do desencontro objeto-relacional, temos que manter um paralelo em alguns aspectos aos bancos de dados relacionais, a exemplo disso temos os OIDs, que são identificadores de objetos.

Como nossos objetos serão salvos em um banco relacional devemos disponibilizar uma forma de encontrá-los. Um OID identifica um único objeto em meio a tantos outros, ele representa a "chave-primária" do nosso banco relacional.

O uso de OIDs facilita a busca de objetos e também o relacionamento entre os objetos. Isso porque quando todas as tabelas utilizam o mesmo tipo de coluna para manter suas chaves, o desenvolvimento de código genérico para lidar com elas é facilitado.

Existem algumas considerações sobre as características de um OID:

• *Um OID não deve ter significado para as regras de negócio:* Existe um fato que pode passar despercebido ao modelarmos nossas tabelas. Qualquer coluna de uma tabela que tem significado para as regras do negócio, pode sofrer mudanças, isso porque as regras de negócio podem mudar. Colocar campos que contém significado como chaves é erro que deve ser evitado. Imaginem a situação onde para identificar uma venda é utilizado como chave primária um número que representa a Nota Fiscal. Inicialmente o número da nota fiscal contém apenas números, porém mudam-se as regras de negócio e agora o número de uma NF pode conter letras também. O tipo da coluna que armazena esse valor deverá ser mudado em todas as tabelas, seja onde é chave primária seja onde é chave estrangeira. Uma simples mudança de tipo torna-se um pesadelo. Sendo assim um OID não deve ter significado algum para os negócios, ele deve ser apenas um número seqüencial que identifica um único objeto em meio a outros;

• *Nível de unicidade de um OID:* Um outro fator que deve ser considerado ao utilizar OIDs está em sua unicidade, ele pode ser único para o universo todo de objetos, ou pode ser único apenas em meio à classe que pertence. Vamos explicar melhor. Você pode ter, por exemplo, um objeto *Aluno* que recebe um OID número 5400. Esse número é único em meio a todos os registros do banco de dados. Não existe outro registro com esse OID. Agora, o número 5400 pode também ser único apenas entre os objetos de tipo *Aluno*, existirão outros registros com esse número, porém pertencerão a outras tabelas/classes. Ao utilizar OIDs deve-se escolher entre essas duas opções, levando em consideração que se escolher a primeira, a coluna que conterá o valor do OID deverá suportar inteiros longos, isso porque você poderá ter bilhões, trilhões de registros em todo o banco de dados.

Mapeamento básico

Ao mapear classes para tabelas, nem sempre uma classe representa uma tabela e vice-versa. É papel do mapeamento indicar o que representa o que, só assim sua camada de persistência saberá para onde salvar seus objetos e de quais tabelas obter os dados que os compõem. Vamos aplicar o mapeamento básico em nosso modelo de classes.

• *Atributos são mapeados para colunas:* Um atributo de uma classe pode ser mapeado para 0 ou mais colunas de uma tabela. Você deve estar se perguntando por que 0 (zero)? Zero porque em uma classe nem todos os atributos são persistidos (salvos em meio físico). Um exemplo disso pode ser uma classe *Aluno* que possui um atributo data de nascimento e idade. *Data de nascimento* é persistido, porém *Idade* não, porque é um atributo calculado, seu valor é obtido de forma dinâmica portanto não precisa ser salvo;

• *Mapeando classes para tabelas:* Em situações simples geralmente classes são mapeadas diretamente para tabelas, exemplo:

```
Classe TDespesa -> Tabela DESPESA
```

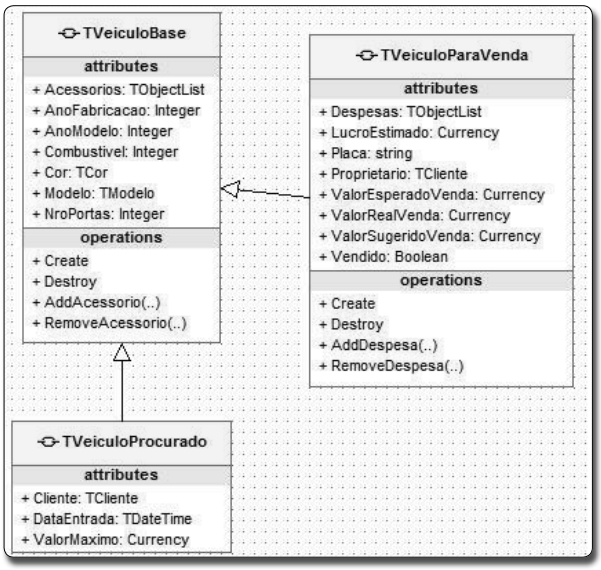


Figura 1. Herança

Porém existem situações mais complexas, por exemplo, quando nossos objetos utilizam herança para alcançar a realidade do negócio. Para demonstrar como mapear uma herança, tomemos por base o diagrama de classes da **Figura 1.**

Existem três formas básicas para se implementar a herança, que é um conceito de orientação a objetos, a um banco de dados relacional:

1. Uma tabela para toda hierarquia de classe: Nessa situação temos uma tabela que armazena os dados de *TVeiculo*, *TVeiculoProcurado* e de *TVeiculoProcurado*. Nesta situação o banco de dados fica menos normalizado, porém aqui um desempenho melhor pode ser obtido já que é menos custoso localizar as informações de um determinado veículo visto que a busca será feita em uma única tabela. Veja a **Figura 2**.

2. Uma tabela por classe concreta da hierarquia: Nessa implementação cada classe concreta terá sua tabela, implicando em problemas quando da inserção de novos atributos comuns às classes da hierarquia. Veja a **Figura 3.**

Se formos inserir um novo atributo à classe base *TVeiculoBase*, por exemplo o atributo *NroRenavam*, uma coluna RENAVAM deve ser criada em ambas as tabelas.

3. Uma tabela por classe: Esta pode ser a melhor forma de implementar herança. O banco de dados fica normalizado e se aproxima mais da realidade do modelo de classes. Veja a **Figura 4.**

Utilizando essa implementação várias tabelas são criadas para representar a hierarquia da herança, isso pode implicar em uma perda relativa de desempenho do banco de dados.

Implementando relacionamentos

1:N

Os relacionamentos em um banco relacional são mantidos através de chaves estrangeiras. Elas permitem relacionar um registro de uma tabela com um registro de uma outra. Veja o diagrama de classes representado na **Figura 5.**

A regra de negócio para o nosso sistema diz que um veículo possui um modelo, e cada modelo pode estar em um ou mais de um veículo. A implementação em nosso banco relacional para essa situação é vista na **Figura 6.** O lado N do relacionamento recebe a chave estrangeira.

M:N

Em outra regra de negócio definimos que um veículo pode possuir acessórios e um acessório pode estar em mais de um veículo. A representação UML para essa situação é vista na **Figura 7**.

O relacionamento anterior em uma estrutura relacional é chamado de M:N. Para implementá-lo temos que introduzir o conceito de tabela associativa. Essa tabela associativa será responsável por manter as referências entre as tabelas envolvidas no relacionamento armazenando as chaves das tabelas envolvidas. Veja a **Figura 8**.

Ao desenvolver um sistema OO onde não temos nenhum banco de dados legado começamos o desenvolvimento pelas classes que envolvem o negócio, conforme fizemos em nosso último artigo. Com nossas entidades prontas construímos então as tabelas baseadas nessas classes, dessa forma o banco relacional pode se adaptar melhor à solução que modelamos em UML. Fazer o contrário, criar as classes baseando-se no banco de dados não é o indicado porque:

1. Conceitos da teoria relacional clássica não são bons na prática – Conceitos como chave composta e chaves com significado não são bem aplicados na prática à orientação a objetos;

2. A estrutura do banco de dados não reflete os requisitos necessários – Os modelos de dados não levam em conta o comportamento esperado, e sim, apenas a forma como os dados serão salvos.

O banco relacional

Seguindo as regras de mapeamento nosso banco de dados tomou a seguinte forma, como visto na **Figura 9**.

Estratégia de persistência

Para salvar os objetos em um banco relacional é preciso aplicar uma estratégia de persistência. Essas estratégias podem ser:

• Utilizar um *framework* de persistência (OPF) existente – Poderíamos aqui utilizar o *InstantObject*, tiOPF ou DePO;

• Criar um OPF próprio – Bem, se temos ótimos *frameworks* criados, criar mais um pode ser muito custoso para o desenvolvimento do sistema e não é algo tão fácil de se implementar;

VEICULO	
ID_VEICULO	INTEGER
ID_MODELO	INTEGER
ID_COR	INTEGER
ANOMODELO	INTEGER
ANOFABRICACAO	INTEGER
COMBUSTIVEL	INTEGER
NUMERO_PORTAS	INTEGER
DATA_ENTRADA	DATE
VALOR_MAXIMO	NUMERIC(15,2)
ID_CLIENTE	INTEGER
LUCRO_ESTIMADO	NUMERIC(15,2)
PLACA	VARCHAR(10)
ID_PROPRIETARIO	INTEGER
VALOR_ESPERADO_VENDA	NUMERIC(15,2)
VALOR_SUGERIDO_VENDA	NUMERIC(15,2)
VALOR_REAL_VENDA	NUMERIC(15,2)
VENDIDO	INTEGER
TIPO_VEICULO	INTEGER

Figura 2. Uma tabela para toda hierarquia

VEICULO_PARA_VENDA	
ID_VEICULO	INTEGER
ID_MODELO	INTEGER
ID_COR	INTEGER
ANOMODELO	INTEGER
ANOFABRICACAO	INTEGER
COMBUSTIVEL	INTEGER
NUMERO_PORTAS	INTEGER
LUCRO_ESTIMADO	NUMERIC(15,2)
PLACA	VARCHAR(10)
ID_PROPRIETARIO	INTEGER
VALOR_ESPERADO_VENDA	NUMERIC(15,2)
VALOR_SUGERIDO_VENDA	NUMERIC(15,2)
VALOR_REAL_VENDA	NUMERIC(15,2)
VENDIDO	INTEGER

VEICULO_PROCURADO	
ID_VEICULO	INTEGER
ID_MODELO	INTEGER
ID_COR	INTEGER
ANOMODELO	INTEGER
ANOFABRICACAO	INTEGER
COMBUSTIVEL	INTEGER
NUMERO_PORTAS	INTEGER
ID_CLIENTE	INTEGER
DATA_ENTRADA	DATE
VALOR_MAXIMO	NUMERIC(15,2)

Figura 3. Uma tabela por classe concreta

• Utilizar o padrão DAO – Aplicar o padrão *Data Access Object* não é muito complicado, temos que entender seus prós e contras.

Nesse curso usaremos DAO.

Entendendo o padrão DAO

O DAO cria uma camada na aplicação, separando as rotinas de acesso a banco de dados, geração de SQL, das rotinas de negócio, de tal modo que os objetos de negócio não saibam como estão sendo gravados. Na **Figura 10** vemos o diagrama de sua estrutura.

Vamos entender um pouco do DAO:

• *BusinessObject:* São os objetos que precisam acessar informações de alguma fonte de dados para realizar suas operações. Em nosso projeto, são as entidades criadas como *TCliente*, *TDespesa*;

• *DataAccessObject:* É encarregado de abstrair e encapsular o acesso a alguma base de dados para disponibilizar um acesso transparente a essa fonte de dados. Cada entidade precisa de seu DAO;

• *DataSource:* Representa a implementação de uma fonte de dados, que pode ser um banco relacional, um banco OO, XML, texto etc.;

• *TransferObject:* Os objetos DAO *podem* utilizar esses tipos de objetos para retornar dados aos *BusinessObject* ou utilizá-los para atualizar um *DataSource*.

Formas de implementação

Para determinar a forma de se implementar um DAO temos que levar em conta os requisitos para nosso *software* no que diz respeito ao banco de dados e tecnologias envolvidas. Podemos estruturar um DAO de forma que seja fácil mudar de SGBD, se esse for o requisito. Para atender essa primeira situação implementamos conforme visto na **Figura 11**.

Observe que para cada tipo de banco de dados temos um DAO específico que será criado através de um padrão *Factory*. Para exemplificar, se fôssemos trabalhar com o *MS SQL Server* e com o *Firebird*, teríamos um *MSSQLDaoFactory* que retornaria os objetos DAO que implementam o acesso ao *MS SQL Server*. Para o *Firebird* seria a mesma coisa.

Para o sistema *DataCar*, trocar de banco de dados não é o requisito principal

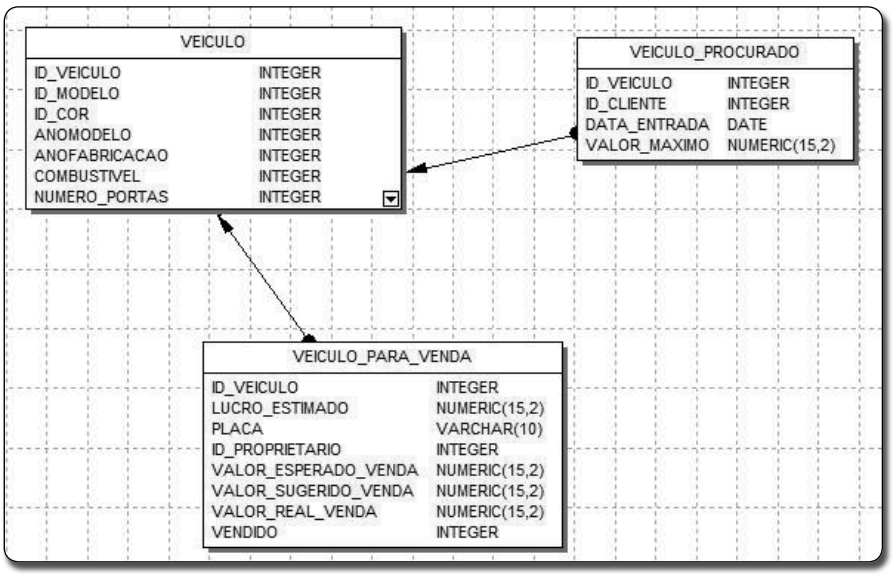


Figura 4. Uma tabela por classe

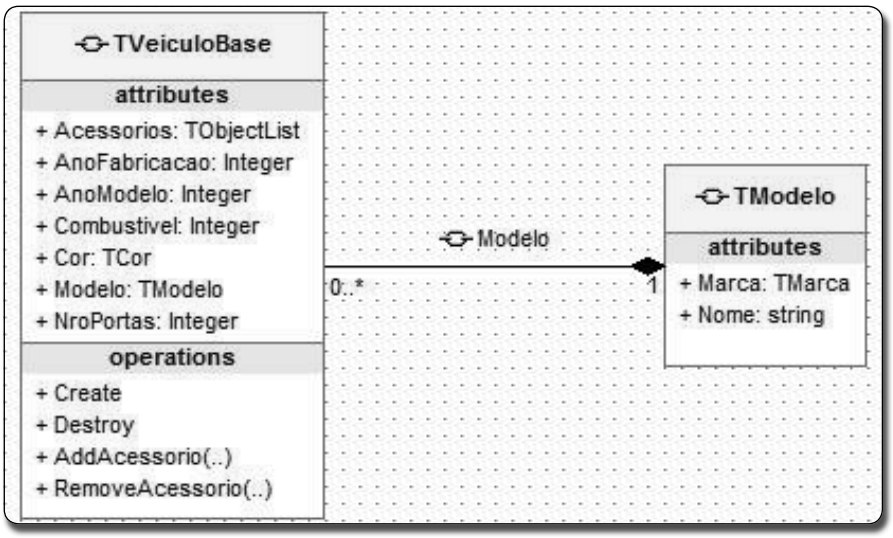


Figura 5. Relacionamento simples entre classes

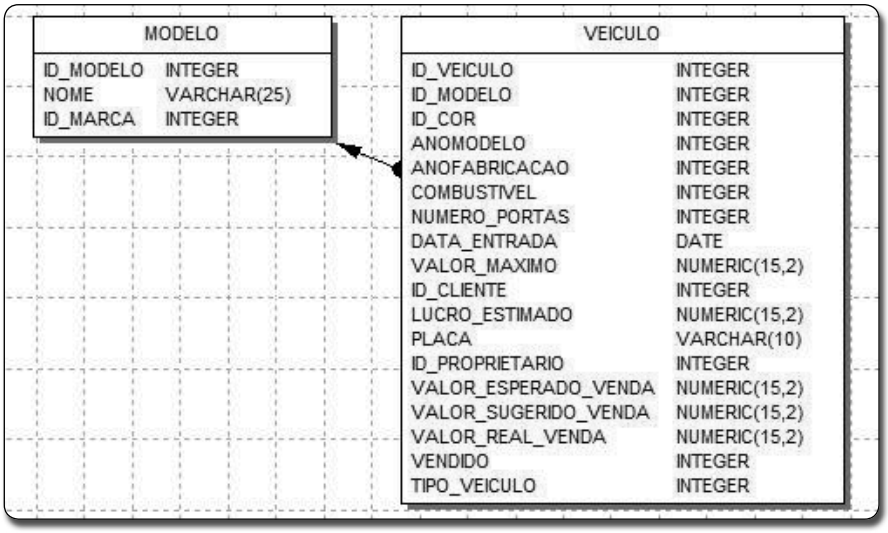


Figura 6. Relacionamento 1:N

porém a forma de acessar o banco de dados pode ser. Inicialmente vamos implementar nosso DAO utilizando o DBX 4, que por sua vez utilizará o *Firebird 1.5*. Caso o desempenho de acesso do DBX 4 não seja satisfatório para nosso sistema, nós poderíamos trocar para outra tecnologia de acesso, por exemplo o FIBPlus ou IBO. Pensando nisso, ao invés de implementarmos o DAO visando uma mudança de SGBD, vamos visar uma mudança de tecnologia de acesso.

Conseqüências de utilização

Prós

• *Transparência:* os objetos clientes podem utilizar os *DataSources* sem saber detalhes específicos de sua implementação, isto porque todo o detalhe de implementação está encapsulado nos objetos DAO;

• *Facilidade de Migração:* a camada DAO permite que uma aplicação possa variar de fabricante de banco de dados de forma facilitada, isto porque toda mudança a ser feita está centralizada da camada DAO;

• *Centralização de Acesso a Dados:* Como toda operação de acesso a dados está delegada aos DAOs, essa camada separa, isola, o restante da aplicação das implementações de acesso a dados. Essa centralização torna a aplicação mais fácil de ser mantida e gerenciada.

Contras

• *Adição de Camada Extra:* o DAO cria uma camada de objetos que ficará entre os objetos cliente e *datasource*. Essa camada precisa ser planejada e implementada. Todas as vantagens de se ter um DAO podem cair diante do fato de ser ter um esforço extra para alcançar os bons resultados, resumindo, teremos mais código.

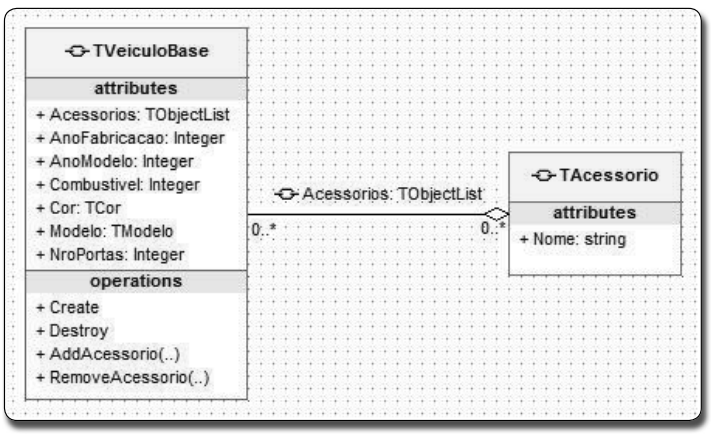


Figura 7. Relacionamento M:N entre classes

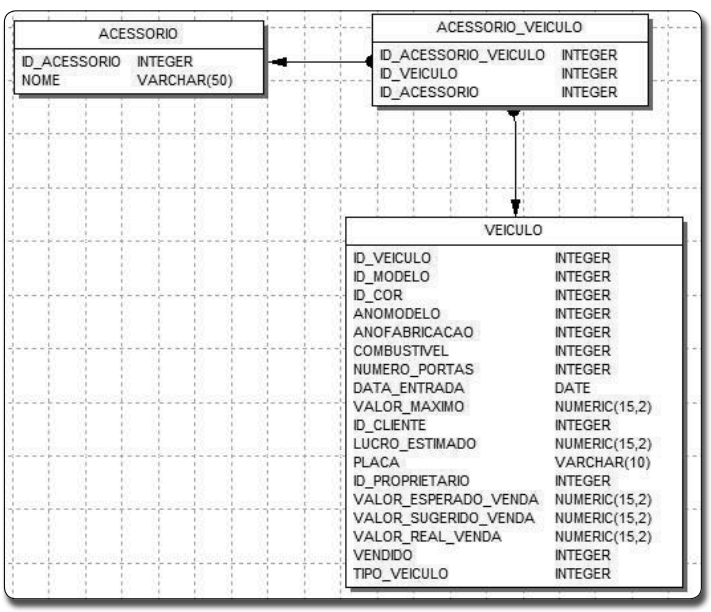


Figura 8. Relacionamento M:N

Colocando a mão na massa

Ao projeto do *DataCar* vamos agora adicionar um *Data Module*. Clicando com o botão direito do mouse sobre o projeto *DataCar2008.exe*, escolha *Add New>Other*. Na janela que surge escolha *Delphi Files* e então *Data Module*. Salve sua *unit* como *dmU.pas* e configure sua propriedade *name* para DM.

Neste novo *DataModule* vamos adicionar um *TSQLConnection*, da guia *dbExpress*. Nele vamos configurar uma conexão com o banco anteriormente criado. Aqui vou utilizar um *driver* de conexão específico para o *Firebird*, desenvolvido pela *CoreLab*. Decidi utilizar o *driver* da *CoreLab* devido um problema que encontrei em utilizar o *driver* padrão para *Interbase* nesta versão 4. Para ser mais exato, o problema ocorre ao tentar salvar um campo *Blob*.

Seguindo o padrão DAO modelei as seguintes interfaces no *ModelMaker*, conforme a **Figura 12**.

Inicialmente cada interface expõe os seguintes métodos:

• Excluir;
• Recuperar;
• Salvar.

> **Nota:** O *ModelMaker* nesta hora faz um *show* a parte em minha opinião. Com nossas interfaces modeladas ele oferece um recurso de automaticamente criar as classes para nós, e depois nos resta apenas implementar os métodos.

Na **Figura 13** vemos as classes criadas pelo *ModelMaker*.

Devido à questão de espaço vou mostrar aqui como a classe *TDaoDbx* e *TMarcaDaoDbx* foram implementadas, lembrando que o código completo até o momento está disponível para *download*.

Como pode ser visto na **Listagem 1** a classe *TDaoDbx* implementa a interface *IDao*. *IDao* não expõe método algum a ser implementado, então na classe criamos os métodos que são necessários para se utilizar o *dbExpress*.

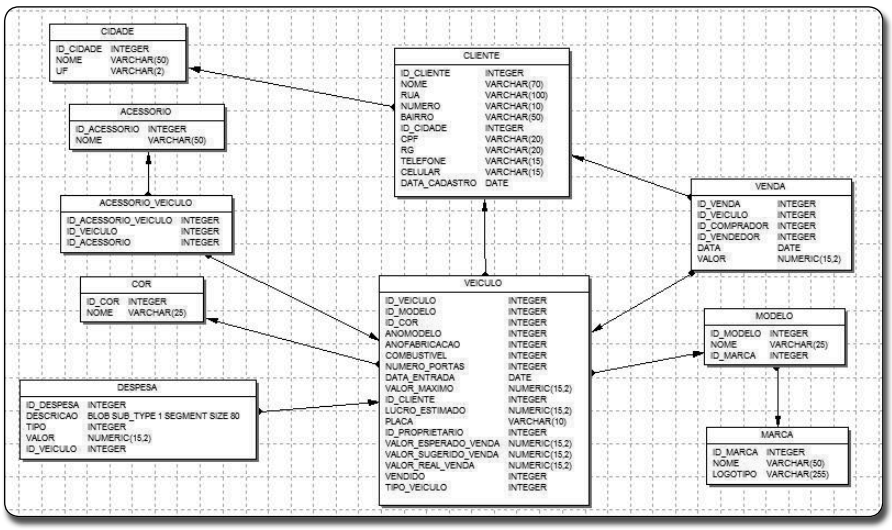


Figura 9. O banco de dados

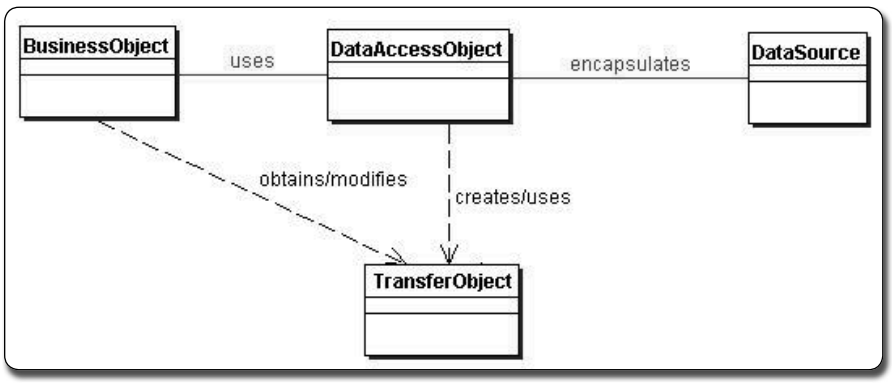


Figura 10. Padrão DAO

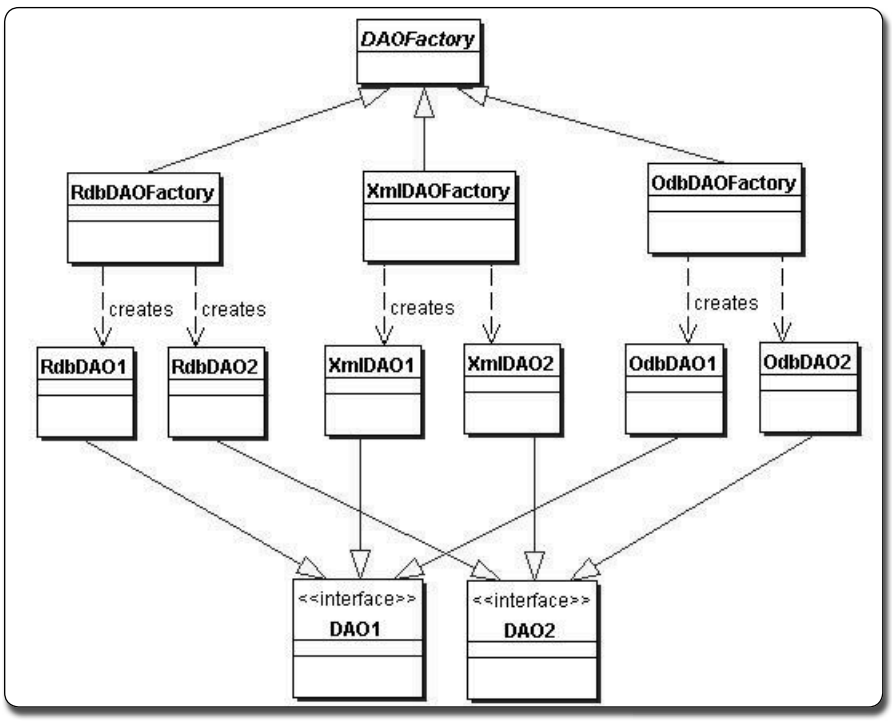


Figura 11. DAO implementado para facilitar troca de SGBD

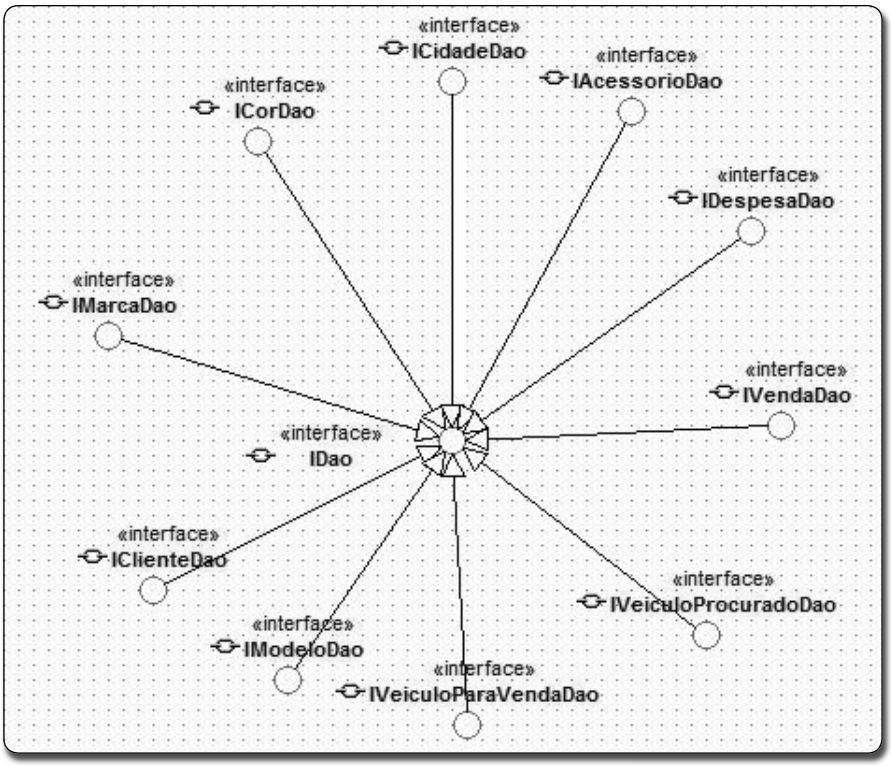


Figura 12. Interfaces DAO

Listagem 1. Classe TDAODBX

```
TDaoDbx = class(TInterfacedObject, IDao)
  private
    SqlQuery: TSqlQuery;

  protected
    procedure AfterConstruction; override;
    procedure BeforeDestruction; override;
    function GeraId(NomeGenerator: string): Integer;
  end;

procedure TDaoDbx.AfterConstruction;
begin
  SqlQuery := TSqlQuery.Create(nil);
  SqlQuery.SQLConnection := dm.ConexaoSQL;
end;

procedure TDaoDbx.BeforeDestruction;
begin
  SqlQuery.Close;
  FreeAndNil(SqlQuery);
end;

function TDaoDbx.GeraId(NomeGenerator: string):
  Integer;
var
  lquery: TSQLQuery;
begin
  result := 0;
  lquery := TSQLQuery.Create(nil);
  lquery.SQLConnection := dm.ConexaoSQL;
  lquery.SQL.Add('SELECT GEN_ID('+ NomeGenerator +',1) AS ID FROM RDB$DATABASE');

  try
    try
      lquery.Open;
      result := lquery.FieldByName('ID').asInteger;
    except

    on e: exception do
      ShowMessage('Generator inexistente: ' + NomeGenerator);
    end;

  finally
    lquery.Close;
    FreeAndNil(lquery);
  end;
end;
```

Temos um objeto do tipo *TSQLQuery* que será utilizado para execução das sentenças SQL e criamos também um método *GeraId* que será responsável pela geração de valores para o ID dos objetos. Veja que estamos criando em tempo de execução um objeto *TSqlQuery*, para receber e as instruções SQL necessárias à execução do sistema.

Na **Listagem 2** temos a implementação da classe DAO responsável pela classe *TMarca*, *TMarcaDaoDbx*. Ela por sua vez implementa *IMarcaDao* e herda de *TDaoDbx*. Observe que os métodos que compõem a persistência,

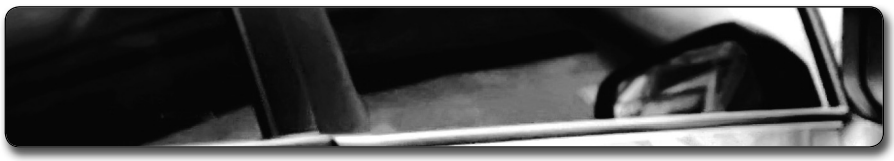
Nota do DevMan

O padrão Factory, de forma simplificada, se encarrega de criar objetos de um determinado tipo sem que quem esteja pedindo para criar saiba o tipo concreto desse objeto. Trazendo isso para código Delphi.

```
procedure TForm1.Button1Click(sender:
TObject);
var
  F: TForm;
begin
  F := FormCadastroClienteFactory.
CreateForm;
  try
    F.ShowModal;
  Finally
    FreeAndNil(F);
  end;
end;
```

TForm1 precisa exibir um determinado formulário de cadastro de cliente então ele solicita à "fábrica de formulário de clientes" que se crie um. Veja que TForm1 não sabe o tipo concreto do formulário que será criado por isso ele irá armazenar o formulário criado em um objeto do tipo TForm. Dessa forma o TForm1 não depende diretamente de nenhum formulário de cadastro de cliente específico. Quem sabe qual tipo que deve ser criado é o FormCadastroClienteFactory.

Recuperar, Salvar, Excluir são fortemente tipados, ou seja, eles esperam por um objeto de tipo específico, no caso por um *TMarca*. Toda a geração de SQL e acesso ao banco ficam centralizadas nas classes DAO.



Testando nosso DAO

Para assegurar que tudo está funcionando vamos implementar alguns testes unitários em nossa classe DAO, dessa forma saberemos se estamos no caminho certo. No *Project Manager*, no projeto de testes vamos adicionar uma nova *unit* e salvá-lo como *MarcaDaoDbxTest.pas*. É preciso também adicionar ao projeto as *units DaoDbx.pas*, *DaoUI.pas e DmUI.pas*.

O teste implementado está na **Listagem 3**. Como foi mostrado na edição anterior, para cada classe que se deseja testar, uma nova classe teste deve ser criada.

Observe que criamos alguns métodos auxiliares, o *LimpaTabela* e o *InsereRegistros*. Ambos tratam de preparar o banco de dados para que possamos realizar os testes. O método *LimpaTabela* é executado no início de cada teste, veja o método *SetUp*.

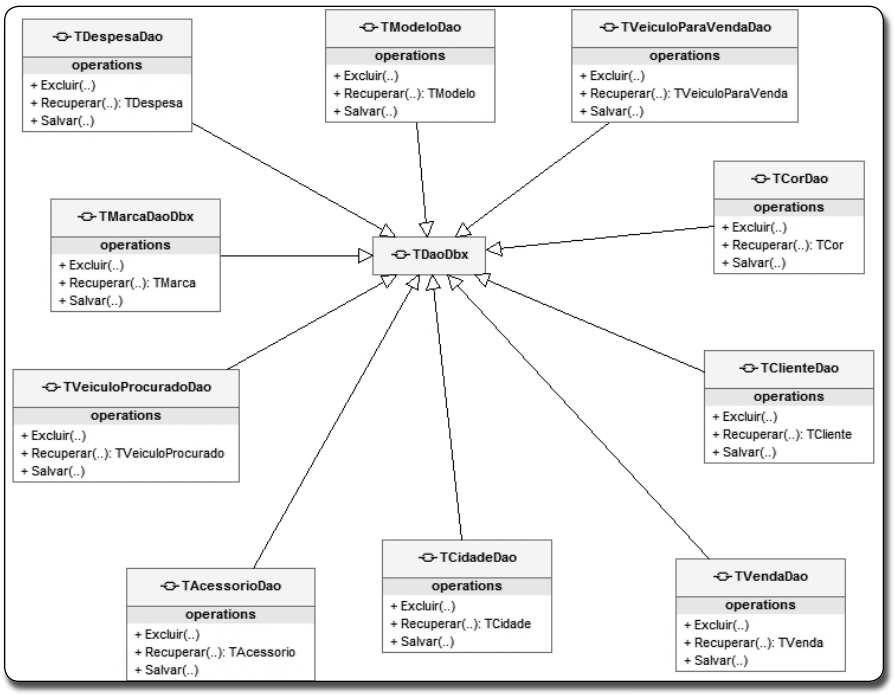
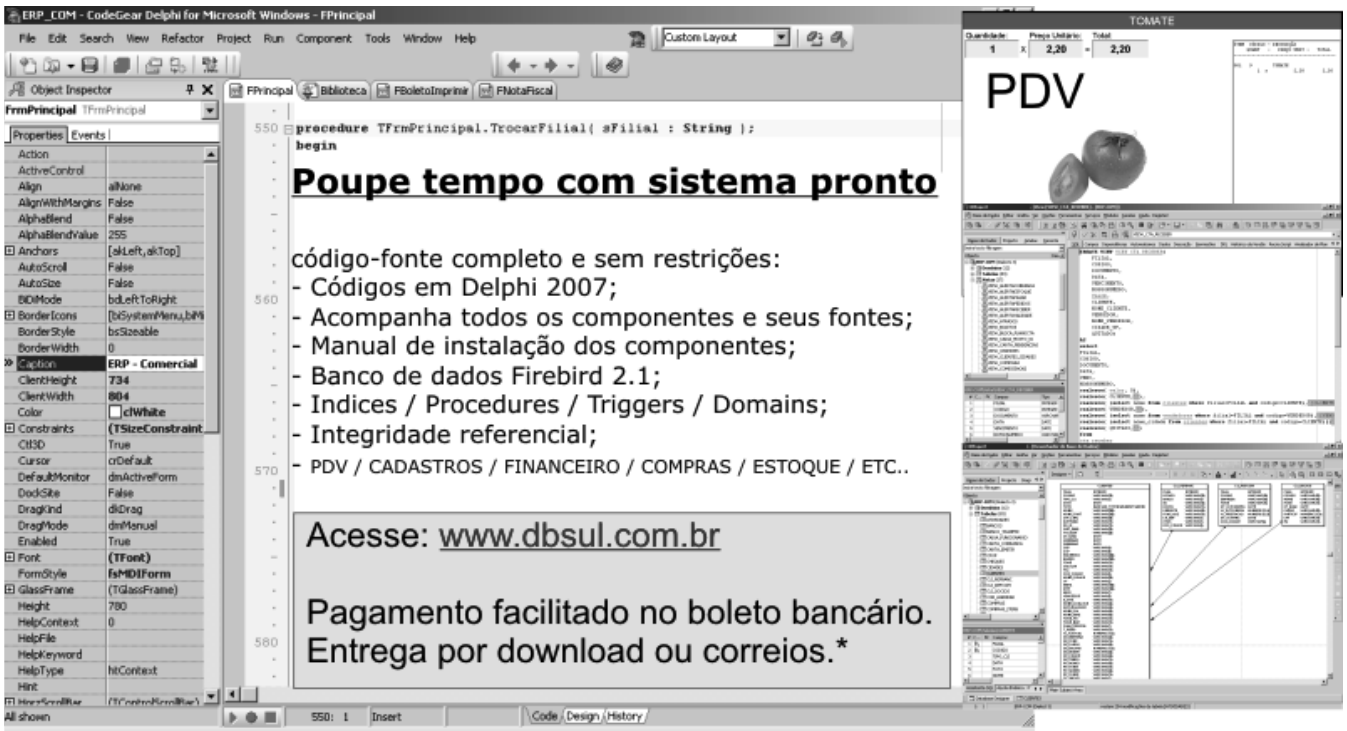


Figura 13. Diagrama de classes DAO



Listagem 2. TMarcaDaoDbx

```
  TMarcaDaoDbx = class(TDaoDbx, IMarcaDao)
  public
    procedure Excluir(const obj: TMarca);
    function Recuperar(Id: Integer): TMarca;
    procedure Salvar(obj: TMarca);
  end;
procedure TMarcaDaoDbx.Excluir(const obj: TMarca);
var
  sql: string;
  resultado: integer;
begin
  sql := 'DELETE FROM MARCA WHERE ID_MARCA = ' + IntToStr(obj.Id);
  try
    dm.ConexaoSQL.ExecuteDirect(sql);
  except
  on e:Exception do
    raise exception.Create('Falha ao excluir Marca: ' + obj.Nome + #13#10 + 'Mensagem
original: ' + e.Message);
  end;
end;
function TMarcaDaoDbx.Recuperar(Id: Integer): TMarca;
var
  Marca: TMarca;
begin
  SqlQuery.Close;
  SqlQuery.SQL.Clear;
  SqlQuery.SQL.Add('SELECT ID_MARCA, NOME, LOGOTIPO FROM MARCA WHERE ID_MARCA = ' +
IntTostr(Id));
  try
    try
      SqlQuery.Open;
      if SqlQuery.IsEmpty then
        Marca := nil
      else
      begin
        Marca := TMarca.Create;
        Marca.Id := SqlQuery.FieldByName('ID_MARCA').AsInteger;
        Marca.Nome := SqlQuery.FieldByName('NOME').AsString;
        Marca.Logotipo := SqlQuery.FieldByName('LOGOTIPO').AsString;
      end;
    except
    on e:exception do
    begin
      ShowMessage('Falha ao recuperar Marca'+ #13+'Mesagem original: '+ e.Message);
      Marca := nil;
    end;
    end;
  finally
    SqlQuery.Close;
    result := Marca;
  end;
end;
procedure TMarcaDaoDbx.Salvar(obj: TMarca);
var
  Sql: string;
  id: Integer;
begin
  if obj.Id = 0 then
    Sql := 'INSERT INTO MARCA VALUES(:ID,:NOME,:LOGOTIPO)'
  else
    Sql := 'UPDATE MARCA ' + 'SET NOME = :NOME, LOGOTIPO = :LOGOTIPO '+
           'WHERE ID_MARCA = ' + IntToStr(obj.Id);
  SqlQuery.Close;
  SqlQuery.SQL.Clear;
  SqlQuery.SQL.Add(sql);
  if obj.Id = 0 then
  begin
    id:= GeraId('GEN_MARCA');
    SqlQuery.Params[0].AsInteger := id;
    SqlQuery.Params[1].AsString := obj.Nome;
    SqlQuery.Params[2].AsString := obj.Logotipo;
  end
  else
  begin
    SqlQuery.Params[0].AsString := obj.Nome;
    SqlQuery.Params[1].AsString := obj.Logotipo;
  end;
  try
    try
      SqlQuery.ExecSQL;
      if obj.Id = 0 then
        obj.Id := id;
    except
    on e:Exception do
    begin
      ShowMessage('Falha ao salvar Marca '+ obj.Nome + #13 + 'Mensagem original: '+
e.Message);
    end;
    end;
  finally
    SqlQuery.Close;
  end;
end;
```

Para testar se a inclusão deu certo criamos um objeto *TMarca* e solicitamos ao DAO que o salve. Então verificamos se o ID do objeto foi alterado.

Quanto ao recuperar, nós primeiramente executamos o método *InsereRegistros*, para que a tabela seja preenchida com algumas informações de teste, então solicitamos ao DAO que nos retorne o objeto de ID 1. Se o retorno for diferente de *nil*, o método funcionou. Depois pedimos ao DAO para retornar um objeto que não existe, de ID 5. Então verificamos se o objeto retornado é *nil*. Se não for o DAO não está funcionando como o esperado.

Para o método excluir passamos um objeto de ID 1 e solicitamos ao DAO para excluí-lo. Executando um *Select* que busca esse ID 1 verificamos se o *TSQLQuery* utilizado está vazio. Se não estiver o DAO não excluiu a marca. Veja o resultado na **Figura 14.**

Conclusão

Separar o acesso ao banco de dados em uma camada dentro da arquitetura

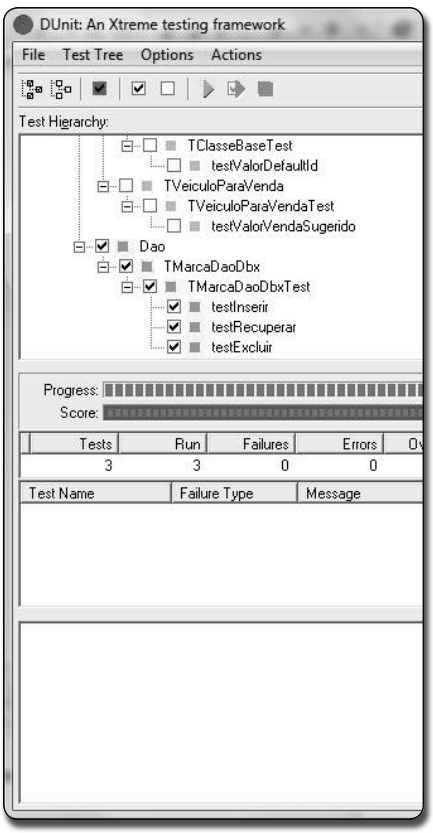


Figura 14. Execução dos testes da classe TMarcaDaoDbx

Listagem 3. Testando TMarcaDaoDbx

```
unit MarcaDaoDbxTest;
interface
uses TestFramework,
     ModelU,
     DaoU,
     DaoDbx,
     dmU,
     SysUtils;
type
  TMarcaDaoDbxTest = class(TTestCase)
  protected
    ldatamodule: Tdm;
    objDao: IMarcaDao;
    procedure SetUp; override;
    procedure TearDown; override;
    procedure LimpaTabela;
    procedure InsereRegistros;
  published
    procedure testInserir;
    procedure testRecuperar;
  end;
implementation
{ TMarcaDaoDbxTest }
procedure TMarcaDaoDbxTest.InsereRegistros;
begin
  ldatamodule.ConexaoSQL.ExecuteDirect('INSERT INTO MARCA VALUES (GEN_ID(GEN_
MARCA,1),''TESTE1'','''')');
  ldatamodule.ConexaoSQL.ExecuteDirect('INSERT INTO MARCA VALUES (GEN_ID(GEN_
MARCA,1),''TESTE2'','''')');
  ldatamodule.ConexaoSQL.ExecuteDirect('INSERT INTO MARCA VALUES (GEN_ID(GEN_
MARCA,1),''TESTE3'','''')');
end;
procedure TMarcaDaoDbxTest.LimpaTabela;
begin
  ldatamodule.ConexaoSQL.ExecuteDirect('DELETE FROM MARCA');
  ldatamodule.ConexaoSQL.ExecuteDirect('SET GENERATOR GEN_MARCA TO 0');
end;
procedure TMarcaDaoDbxTest.SetUp;
begin
  inherited;
  objDao := TMarcaDaoDbx.Create;
  ldatamodule := Tdm.Create(nil);
  ldatamodule.ConexaoSQL.Open;
  LimpaTabela;
end;
procedure TMarcaDaoDbxTest.TearDown;
begin
  inherited;
  ldatamodule.ConexaoSQL.Close;
  FreeAndNil(ldatamodule);
end;
procedure TMarcaDaoDbxTest.testExcluir;
var
  Obj: TMarca;
  SqlQuery: TSQLQuery;
begin
  Obj := TMarca.Create;
  Obj.Id := 1;
  InsereRegistros;
  objDao.Excluir(Obj);
  SqlQuery := TSQLQuery.Create(nil);
  SqlQuery.SQLConnection := dm.ConexaoSQL;
  SqlQuery.SQL.Add('SELECT ID_MARCA FROM MARCA WHERE ID_MARCA = 1');
  SqlQuery.Open;
  check(SqlQuery.IsEmpty,'O select deveria retornar vazio');
  SqlQuery.Close;
  FreeAndNil(SqlQuery);
  FreeAndNil(Obj);
end;
procedure TMarcaDaoDbxTest.testInserir;
var
 Obj: TMarca;
begin
  Obj := TMarca.Create;
  Obj.Nome := 'FIAT';
  Obj.Logotipo := '';
  objDao.Salvar(Obj);
  check(Obj.Id > 0,'O id deveseria ser maior que zero');
  FreeAndNil(obj);
end;
procedure TMarcaDaoDbxTest.testRecuperar;
var
 obj: TMarca;
begin
  obj := nil;
  InsereRegistros;
  obj := objDao.Recuperar(1);
  check(obj <> nil,'o obj deveria ser diferente de nil');
  check(obj.Nome = 'TESTE1','o nome deveria ser igual a "TESTE1"');
  obj := objDao.Recuperar(5);
  check(obj = nil,'o obj deveria ser igual a nil');
  FreeAndNil(obj);
end;
initialization
  RegisterTest('Modelo/Dao/TMarcaDaoDbx', TMarcaDaoDbxTest.Suite);
end.
```

de nossos sistemas, exige um certo trabalho a mais, porém esse esforço é compensado com a facilidade de manutenção obtida. Veja que conseguimos realizar testes até mesmo em nossa camada de dados. Programando do "jeitão" tradicional, com SQL nos formulários, você conseguiria testar tudo isso de forma automática como fizemos aqui? Fica aqui a minha pergunta a você leitor.

Para a próxima edição vamos unir nossas entidades e nossa camada de dados através de uma camada de interface de usuário. Continue acompanhando. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução à API

Entenda e saiba como utilizar Interface de Programação de Aplicativos (API) em suas aplicações

Resumo DevMan

Neste artigo aprenderemos o que são as Interfaces de Programação de Aplicativos (API'S), como usá-las no dia-a-dia. Veremos também como identificar uma API e por onde começar para desenvolver usando-as.

Faremos alguns exemplos práticos para exercitar o aprendizado.

Nesse artigo veremos

- O que são API's;
- Como usá-las na prática;
- Para que servem.

Qual a finalidade

- Entender ainda melhor o que se passa pelos bastidores de nossos sistemas.

Quais situações utilizam esses recursos?

- Qualquer tipo de software está sujeito ao uso de API's, seja para controlar um determinado hardware, seja para ajustar algum software.

Adriano Santos

falecom@adrianosantos.pro.br

é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Mantém o blog Delphi to Delphi (www.delphi-todelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

A grande vantagem em ser um profissional de programação, a meu ver, é que acabamos, involuntariamente, conhecendo a fundo o sistema operacional para o qual desenvolvemos. Sou capaz até de afirmar que o principal requisito para ser um bom programador é, certamente, conhecer ao máximo todas as características e funcionamento do S.O. Isso porque um dos principais fatores para se desenvolver um bom aplicativo está na sua boa integração com a plataforma onde será executado.

Softwares desenvolvidos em *Delphi* têm como plataforma alvo o sistema operacional da *Microsoft*: o *Windows*. Quando desenvolvemos em qualquer versão do *Delphi* já estamos, de certa forma, compatibilizando nosso sistema com o *Windows*. Mas às vezes nos deparamos com situações inusitadas que nos levam a entender o funcionamento mais detalhado do S.O. para que a aplicação rode corretamente.

Também há situações em que precisamos de certa ajuda da plataforma para realizar determinadas tarefas, como saber o tamanho de um arquivo ou usar o gerenciador padrão de envio de e-mail, por exemplo.

Neste artigo veremos como funcionam as API's. Elas nos auxiliam e realizam

as mais diversas tarefas, desde as mais simples até as mais e complexas e demoradas. Entenderemos o que são, para que servem e como usá-las. Aprenderemos que, além do Windows, outros programas também são dotados de API's.

O que são API's?

Interface de Programação de Aplicativos (API), é um conjunto de instruções e rotinas de um determinado *software* que se faz externo para que outras aplicações façam uso dele. Em outras palavras, um aplicativo pode expor um conjunto de rotinas que podem ser utilizadas por qualquer linguagem de programação. Essas API's são funções disponíveis apenas para uso em programação e não são divulgadas ou utilizadas por usuários. Elas permitem, dentre tantas coisas manipular informações do aplicativo, criar janelas (no caso do *Windows*), acessar arquivos, criptografar dados etc.

Quando abrimos o *Delphi*, um novo formulário e, consequentemente, uma nova aplicação é gerada. O formulário criado já possui as características básicas do sistema operacional. Por exemplo: o conjunto de botões *Minimizar, Maximizar/Restaurar* e *Fechar*, já faz parte do sistema operacional e é controlado por uma API. Desta forma, todos os nossos formulários possuem as mesmas características. Isso significa que usamos API e nem mesmo sabemos. Além do *Windows* podemos citar outros exemplos de *softwares* que possuem API's como o *BDE Administrator*, *software* usado para conexão a bancos de dados.

Se entendermos e soubermos como manipular API's de um determinado programa, podemos simplesmente ter mais controle da situação em determinados momentos, ou apenas, facilitar nosso trabalho.

Construindo exemplos

Para exemplificar e entendermos ao certo como utilizar API's, vamos construir uma série de exemplos e ver seu funcionamento. De início vejamos como fazer nossos formulários abrirem de forma animada em nossas aplicações.

Abra o *Delphi* e crie uma nova aplicação usando *File>New>Application* ou utilize a aplicação gerada pelo *Delphi*. Salve a *Unit* principal como *uPrincipal.pas* e o projeto como *Animacoes.dpr*. Adicione um novo formulário à aplicação usando para isso o *menu File>New>Form* e salve-o como *uCalendario.pas*. Em *uPrincipal* insira um componente *TRadioGroup*(*rdgOpcoes*). Nele insira os itens *AW_BLEND*, *AW_CENTER*, *AW_HOR_POSITIVE*, *AW_HOR_NEGATIVE*, *AW_VER_POSITIVE* e *AW_VER_NEGATIVE*, todos na propriedade *Items* um por linha. Modifique a propriedade *Columns* para 3. Arraste um componente *Button* com o *Caption* igual a "Abrir formulário" para o formulário e um *SpinEdit*(*SpinEdit1*) (**Figura 1**).

Pressione *Alt + F11* e adicione ao formulário principal o segundo formulário que criamos. O objetivo é fazer com que a segunda tela abra sempre com uma animação. Utilizaremos a API *AnimateWindow* que é responsável por animar as telas do *Windows*. Um dos recursos dessa API já é conhecido por muitos usuários. Ele chama-se *Esmaecimento*. Quando abrimos uma janela no *Windows XP*, através do *menu Iniciar*, se repararmos bem ao clicar no item o menu clicado some gradativamente, porém é tão rápido que mal percebemos. Isso é o efeito *Esmaecimeto*. Para ativar ou desativar o recurso, basta ir ao *Painel de Controles>Vídeo>Aparência>Efeitos*. O primeiro item diz respeito a isso. Veja na **Figura 2**.

No meu caso minha cópia do *Windows* está em inglês, mas veja o primeiro item. Está marcado como *Esmaecimento*(*Fade*

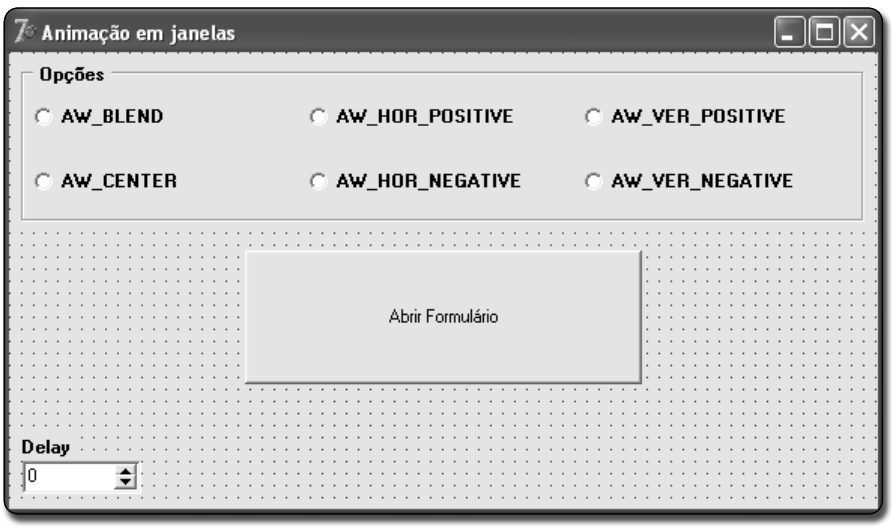


Figura 1. Exemplo de tela para animação

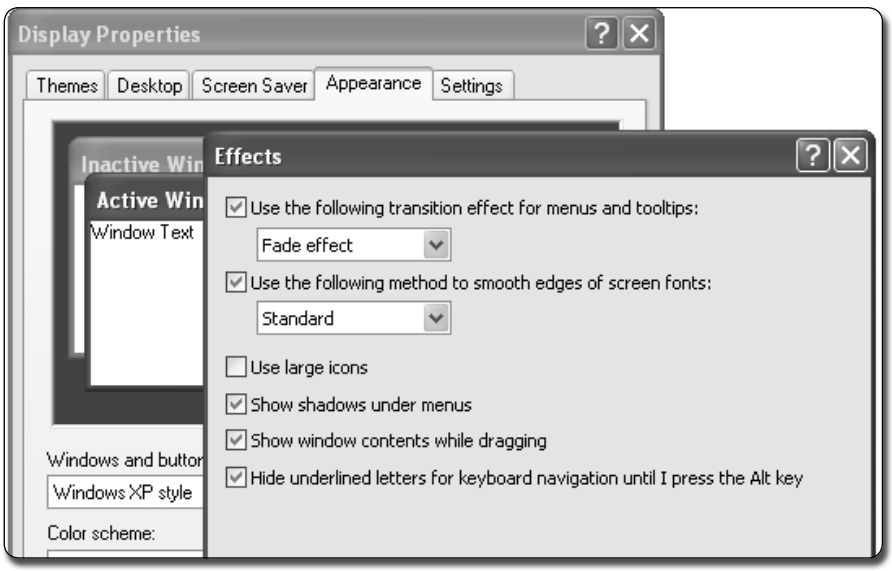


Figura 2. Efeito esmaecimento

effect). Isso significa que os *menus* em todos os aplicativos terão um efeito de esmaecimento sempre que clicados. Vamos simular esse efeito usando o *Delphi* através da função da API *AnimateWindow* como mencionado anteriormente.

Dê um duplo clique no botão que colocamos em tela e codifique conforme a **Listagem 1**. O que estamos fazendo é chamar a função *AnimateWindow* passando como parâmetro qual o *Hadle* do objeto que será animado, o tempo da animação e o tipo dela. Pressione *F9* na aplicação e selecione a opção *AW_BLEND.* Coloque um tempo em *milisegundos*, pressione o botão no meio da tela e veja o efeito.

A **Listagem 1** é bem simples de entender, então vamos lá. O método *AnimateWindow* é o principal personagem de nosso exemplo. Estamos fazendo o *case..of* usando como condicional a propriedade *ItemIndex* do *RadioGroup*, assim, dependendo o item selecionado, enviamos uma constante para o último parâmetro da função, que é justamente a forma de animação que desejamos. As opções de animação são:

- *AW_BLEND:* Efeito esmaecido. O formulário surge gradativamente na tela;
- *AW_CENTER:* O formulário surge do centro para as bordas da janela;
- *AW_HOR_POSITIVE:* A janela surge da esquerda para a direita;
- *AW_HOR_NEGATIVE:* A janela surge da direita para a esquerda;
- *AW_VER_POSITIVE:* A janela surge de cima para baixo;
- *AW_VER_NEGATIVE:* A janela surge de baixo para cima.

As opções se parecem muito com os efeitos de transição do conhecido *software* de apresentações da *Microsoft*, o *PowerPoint*. Além do tipo de animação, ainda temos que informar o *Handle* do formulário no primeiro parâmetro e o *delay* (tempo de abertura) no segundo parâmetro. Nesse caso usamos a propriedade *Value* do *SpinEdit1*.

Execute a aplicação e efetue os testes selecionando os itens do *RadioGroup*, modificando o *SpinEdit* e clicando no botão. Veja que para cada item selecionado uma animação é executada ao abrir o formulário.

> **Nota**: *Handle* é uma espécie de identificador único do objeto que está sendo criado. O *Windows* atribui um número a cada objeto criado na memória, ou seja, toda vez que nossa aplicação é aberta um novo número é atribuído a cada controle de tela. Esse número é chamado de *Handle*. Isso é feito também para outros programas do *Windows*, como *Calculadora*, *Bloco de Notas* etc. Pelo *Handle* o sistema operacional consegue saber exatamente quem é o objeto, seja um botão ou formulário.

Muito bem, mas como saber se estou utilizando uma API ou simplesmente uma função do próprio *Delphi*? Bastante fácil, experimente retornar ao *Delphi*, pressionar *Ctrl* e clicar com o botão esquerdo do mouse sobre o método *AnimateWindow*. Ao fazer isso o *Delphi* localizará a declaração dessa função/procedimento e abrirá a *Unit* onde ela encontra-se declarada. Faça isso e veja o que acontece. A *Unit Windows* é aberta e aponta para a declaração:

```
function AnimateWindow;
  external user32 name 'AnimateWindow';
```

Aliás, podemos ver que há dezenas ou talvez centenas de declarações de funções nessa *Unit*. Isso mesmo, todas elas são API's do *Windows*, cada uma com sua devida finalidade. Repare a palavra *user32* seguida pela palavra reservada *external*. *User32* é a *Dll* que possui a função *AnimateWindow* e *external* indica que a função é externa, ou seja, está alocada fora do *Delphi*, nesse caso em uma *Dll*. Experimente abrir o diretório *C:\Windows\System32* e procurar por *user32.dll* e verá que esse arquivo existe.

O mais interessante de tudo isso é que a função que usamos não está sozinha, existem dezenas de outras funções e procedimentos declarados na *dll*, é só observar a *Unit Windows*. Veja atentamente as funções a seguir:

- *SetFocus:* Envia o foco para o controle;
- *MessageBox:* Exibe mensagens na aplicação.

Com certeza as duas funções você já deve ter utilizado em seus sistemas. A primeira delas é usada para enviar o foco, o cursor em si, para o controle desejado, por exemplo:

```
SpinEdit1.SetFocus;
```

Pois bem, usando esse método estamos enviando o cursor para dentro do componente *Spin* com o nome *SpinEdit1*. Estamos usando a API do *Windows* para fazer isso. Vamos analisar a função de perto? Então, volte ao *Delphi*, entre no código do botão e digite em qualquer lugar a instrução anterior. Agora pressione *Ctrl* e clique sobre o método *SetFocus*. Repare que somos redirecionados diretamente para a *Unit Ctrls* e certamente deve estar achando que me enganei, mas estamos no caminho certo.

Listagem 1. Código de chamada ao formulário com animação

```
procedure TuPrincipal.Button1Click(Sender: TObject);
begin
  frmFormulario2.BringToFront;
  case rdgOpcoes.ItemIndex of
    0: AnimateWindow(frmFormulario2.Handle, SpinEdit1.Value, AW_BLEND);
    1: AnimateWindow(frmFormulario2.Handle, SpinEdit1.Value, AW_CENTER);
    2: AnimateWindow(frmFormulario2.Handle, SpinEdit1.Value, AW_HOR_POSITIVE);
    3: AnimateWindow(frmFormulario2.Handle, SpinEdit1.Value, AW_HOR_NEGATIVE);
    4: AnimateWindow(frmFormulario2.Handle, SpinEdit1.Value, AW_VER_POSITIVE);
    5: AnimateWindow(frmFormulario2.Handle, SpinEdit1.Value, AW_VER_NEGATIVE);
  end;
  frmFormulario2.show;
end;
```

Listagem 2. Código do SetFocus

```
procedure TWinControl.SetFocus;
var
  Parent: TCustomForm;
begin
  Parent := GetParentForm(Self);
  if Parent <> nil then
    Parent.FocusControl(Self)
  else if ParentWindow <> 0 then
    Windows.SetFocus(Handle)
  else
    ValidParentForm(Self);
end;
```

Observe o código da **Listagem 2** com o código completo da função *SetFocus*.

Caso seja um controle de tela, a função *FocusControl* é chamada e uma série de outras funções são executadas chamando no final dos processos a *function SetFocus*. Caso contrário *SetFocus* é chamada diretamente. No final das contas, quem acaba realmente realizando o trabalho de focar no componente é a função própria *SetFocus* presente na *Unit Windows*. Em suma, estamos usando API sem nem saber, como havia dito no início do artigo. Vamos a outro caso.

Executando ou abrindo outros programas

Às vezes precisamos abrir arquivos de outros programas, ou mesmo, abrir sistemas diretamente de nossa aplicação. Já perdi as contas de quantas vezes meus clientes pediram para incluir um atalho no sistema para abrir a *Calculadora* do *Windows* ou mesmo o *MS Word*. Pois bem, como fazer isso? Lembram-se da *Unit Windows* cheia de funções da API do *Windows*? Pois a solução estava lá, novamente. Porém desta vez usaremos outra API, o *Kernel32.dll*.

A *dll Kernel32.dll* é uma das principais *dlls* do sistema *Windows*. Ela possui uma vasta quantidade de funções e o próprio *Windows* a usa para executar determinadas operações. Através dela e de sua ajudinha, vamos ver como abrir um programa em nosso sistema.

Adicione um novo botão no formulário principal e chame-o de *btnCalculadora*. Modifique seu *Caption* para *Calculadora* e logo em seguida clique duas vezes nele para acessarmos o código-fonte. O código é bem simples, apenas digite:

```
WinExec('C:\Windows\System32\Calc.exe',
  SW_SHOWNORMAL);
```

Perceba que é bem simples executar a calculadora usando o *WinExec*. O primeiro parâmetro é o programa que será executado com diretório, unidade de disco e tudo, ou seja, o caminho completo. O outro serve para indicarmos como o arquivo/programa deve ser aberto. Alguns dos possíveis valores são:

- *SW_SHOWMINIMIZED:* Abre a aplicação Minimizada;
- *SW_SHOWMAXIMIZED:* Abre a aplicação Maximizada.

Se não tiver certeza de que *WinExec* pertence à API, certifique-se clicando na função com a tecla *Ctrl* ativada. Veja que o *Delphi* te enviará para a *Unit Windows* e a declaração da função é a seguinte:

```
function WinExec;
  external kernel32 name 'WinExec';
```

Olhe a palavra reservada *external* novamente e apontando para *kernel32*. Execute a aplicação e veja que ao clicar no botão, a *calculadora* é aberta (**Figura 3**).

Bem, vimos que funciona perfeitamente, mas será que a função *WinExec* é capaz de abrir um arquivo caso seja necessário? E se precisássemos abrir o arquivo texto no *Bloco de Notas* para fazer qualquer edição? Vamos testar? Abra o *Bloco de Notas*, digite um texto qualquer e salve o arquivo com o nome *Teste.txt* no diretório onde salvou este projeto. Em seguida volte ao código-fonte do botão *Calculadora* e modifique o comando *WinExec* como a seguir:

```
WinExec('C:\Teste.txt', SW_SHOWNORMAL);
```

Não deixe de observar o caminho e nome do arquivo. Execute a aplicação e clique no botão. Veja que nenhuma ação é executada, isso porque *WinExec* é usada apenas para abrir programas. Para arquivos devemos recorrer a outra função da API, a *ShellExecute*. Volte mais uma vez ao código e desta vez apague a função *WinExec* e escreva no lugar a linha a seguir:

```
  ShellExecute(0, 'OPEN', 'Teste.txt', '',
'C:\', SW_SHOWNORMAL);
```

Eu também concordo que o tamanho da função é maior, pois pede mais parâmetros. O primeiro deles é o *Handle* que estamos passando como *0*. Em seguida devemos informar qual o comando que realizaremos. Aqui estamos usando *OPEN*. Então informamos o arquivo a ser aberto, seus parâmetros ("usamos vazio"), o diretório onde se encontra armazenado e por fim como a janela será aberta. Efetue o teste e veja o que acontece (**Figura 4**). Se quiser conferir, veja sua declaração a seguir:

```
function ShellExecute; external shell32 name
'ShellExecuteA';
```

ShellExecute é uma função da *Dll Shell32.dll*. O processo de aprendizado não é tão ruim assim como parecia, isso posso afirmar. Vamos a outros exemplos.

Interagindo com o teclado

Quem está acostumado a usar aplicativos como *Excel e Word*, sabe que na barra de status de tais aplicativos conseguimos uma série de informações. Algumas delas

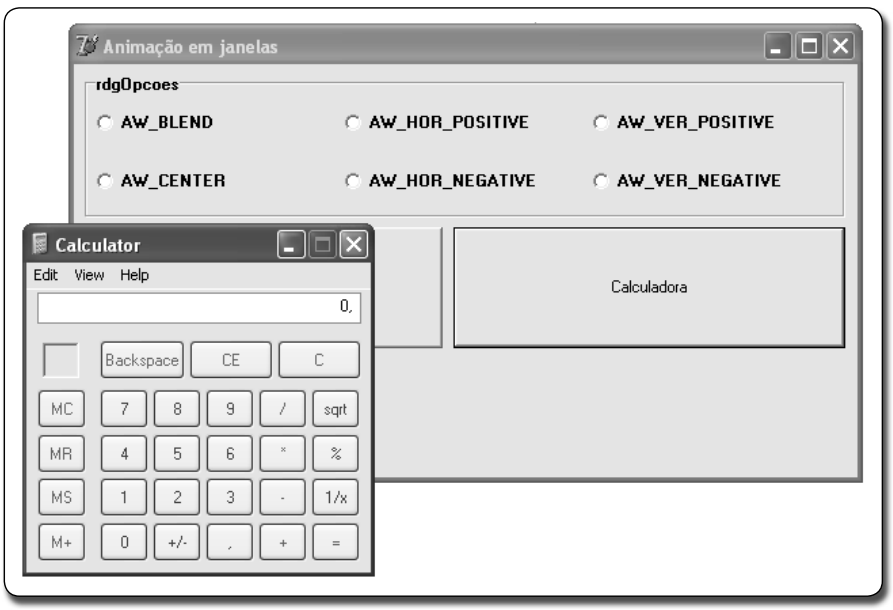


Figura 3. Calculadora aberta pela aplicação

Listagem 3. Código para ativação/desativação das teclas

```
procedure TForm1.LigaDesligaLuzesTeclado(Tecla: Word; AOn: Boolean);
var
  Estado: TKeyboardState;
begin
  GetKeyboardState(Estado);
  if Boolean(Estado[Tecla]) <> AOn then
  begin
    keybd_event(Tecla, MapVirtualKey(Tecla, 0), KEYEVENTF_EXTENDEDKEY, 0);
    keybd_event(Tecla, MapVirtualKey(Tecla, 0), KEYEVENTF_EXTENDEDKEY or KEYEVENTF_KEYUP, 0);
  end;
end;
```

Listagem 4. Código do controle Timer

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if Odd(GetKeyState(VK_CAPITAL)) then
  begin
    StatusBar1.Panels[0].Text := 'CAPS';
    ckCaps.Checked := True;
  end
  else
  begin
    StatusBar1.Panels[0].Text := '';
    ckCaps.Checked := False;
  end;
  if Odd(GetKeyState(VK_NUMLOCK)) then
  begin
    StatusBar1.Panels[1].Text := 'NUM';
    ckNum.Checked := True;
  end
  else
  begin
    StatusBar1.Panels[1].Text := '';
    ckNum.Checked := False;
  end;
end;
```

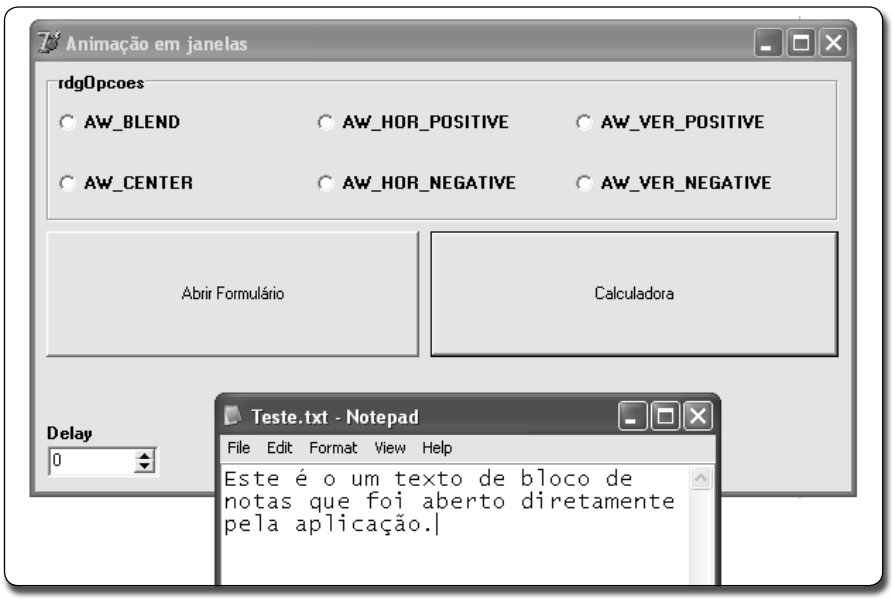


Figura 4. Bloco de Notas chamado pela aplicação

dizem respeito ao estado de algumas teclas, como *CapsLock* e *NumLock*, por exemplo. Poucos sabem que por trás desses aplicativos existe uma série de API's rodando para ajudar a realizar tarefas. E são justamente essas tarefas que quero mostrar para que possamos entender ainda melhor como funcionam as API's.

Construa uma aplicação nova usando como base o *layout* da **Figura 5**. Veja que incluí um *GroupBox* sem um nome específico e dentro dele dois *CheckBoxes*(*ckCaps* e *ckNum*). Usei como *Caption* "*CapsLock*" e "*NumLock*", respectivamente.

Incluí também um componente *StatusBar* adicionando a ele dois painéis. Para isso, clique duas vezes sobre o controle e clique no botão *Add New* ou pressione *Insert*(*Ins*) no teclado. Arraste também um *Timer*.

O exemplo consiste em mostrar nos painéis *0* e *1* do *StatusBar*, o estado das teclas *CapsLock* e *NumLock*, assim o usuário de nosso sistema poderá ver se as teclas estão ativadas ou desativadas.

Ao abrir o aplicativo ele fará a verificação das teclas e mostrará as mensagens *CAPS* para *CapsLock* ligado e *NUM* para *NumLock* ligado. Também atualizará os

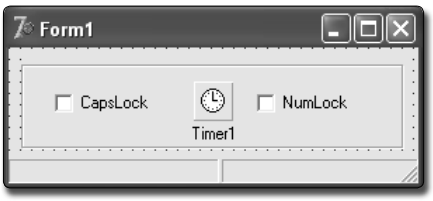


Figura 5. Exemplo de tela para controle de teclas

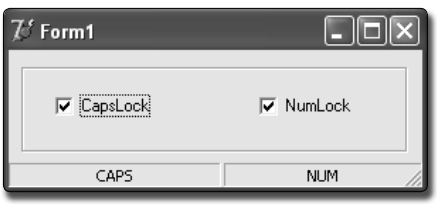
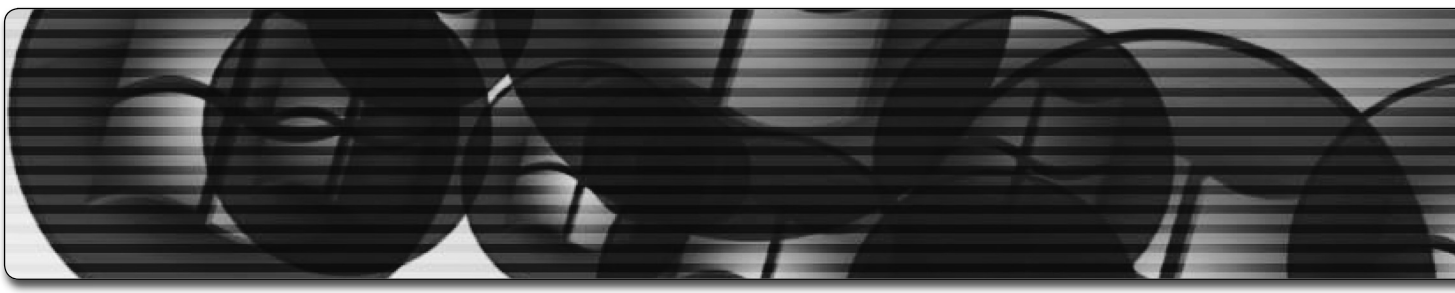


Figura 6. Aplicação acusa o estado das teclas CAPS e NUM LOCK



dois *CheckBoxes* na parte superior. Pressionando os referidos *CheckBoxes* ligaremos e desligaremos as teclas automaticamente.

Usaremos para isso as funções *keybd_event*, *GetKeyboardState* e *MapVirtualKey* da API do *Windows*. Ambas pertencem à API *User32.dll* que já conhecemos. Veja suas declarações a seguir:

```
function GetKeyboardState;
  external user32 name 'GetKeyboardState';
procedure keybd_event;
  external user32 name 'keybd_event';
function MapVirtualKey;
  external user32 name 'MapVirtualKeyA';
```

A primera, *GetKeyboardState*, é responsável por receber o estado atual da tecla, enquanto que *SetKeyboardState* faz o contrário, ou seja, configura um novo estado. Por fim, *MapVirtualKey* faz o trabalho de mapeamento das teclas para descobrir *quem é quem* em tempo de execução. Vejamos como utilizá-las na prática. Encontre a sessão *Public* do nosso exemplo e declare uma nova *procedure* como podemos ver a seguir:

```
procedure LigaDesligaLuzesTeclado(Tecla: Word;
          bOn: Boolean);
```

Essa *procedure* será responsável por ligar ou desligar as teclas *CapsLock* e *NumLock*. Veja que nosso procedimento possui dois parâmetros apenas. O primeiro recebe a constante que representa a tecla, em nosso caso usaremos *VK_CAPITAL* para o *CapsLock* e *VK_NUMLOCK* para a *NumLock*. O parâmetro é do tipo *Word* porque na verdade as teclas são representadas por números no *Delphi*, veja trecho da *Unit Windows*:

```
...
const
  VK_CAPITAL = 20;
  VK_NUMLOCK = 144;
...
```

O segundo parâmetro informa se precisamos ativar (*True*) ou desativar (*False*). Pressione *Ctrl + Shift + C* para que o *Delphi* crie o cabeçalho para nós. Em seguida digite o código da *procedure* conforme a **Listagem 3.**

A primeira coisa que fazemos no código é receber o estado atual da tecla no *Array Estado*, que pode ser ligado ou desligado. Quem é responsável por isso é a API *GetKeyboardState* e ela recebe como parâmetro a variável *Estado* do tipo *TKeyboardState*. Em seguida verificamos se o *Estado* da tecla é diferente do *Estado* recebido no segundo parâmetro. Caso seja, então invertemos o valor usando a API *keybd_event* em conjunto com *MapVirtualKey* passando a tecla para ela.

Clique duas vezes no *CheckBox ckCaps* e digite o código a seguir:

```
LigaDesligaLuzesTeclado(
  VK_CAPITAL, ckCaps.Checked);
```

Repita o passo anterior para o outro *CheckBox*, mas desta vez cole o código que segue:

```
LigaDesligaLuzesTeclado(
  VK_NUMLOCK, ckNum.Checked);
```

Perceba que ambos os *CheckBoxes* chamam a procedure *LigaDesligaLuzesTeclado* apenas diferenciando qual a tecla que será ativada/desativada. Antes de executar a aplicação, vamos então montar as instruções para detecção automática do estado das teclas. Clique duas vezes no *Timer* e digite o código da **Listagem 4.**

Vamos entender o código. Estamos utilizando o método *Odd* para descobrir se o resultado da função *GetKeyState* é ímpar ou par. Se par, então o número é 0, ou seja, desligado. Caso contrário significa que a tecla está ligada, então exibimos o valor *CAPS* o *NUM*, dependendo da função. Além do *Odd* estamos usando outra novidade, a API *GetKeyState*. Já deu para entender o que ela retorna certo? Pois bem, é outra API, seja sua declaração:

```
function GetKeyState; external user32 name
'GetKeyState';
```

Agora sim, compile e teste a aplicação. Veja que tudo funciona perfeitamente (**Figura 6**). O mais interessante de tudo isso é que usamos recursos do próprio *Windows* sem a necessidade de desenvolver uma grande solução. Essas são algumas das vantagens em se conhecer bem o Sistema Operacional.

Conclusão

Vimos de forma sucinta, como desenvolver aplicações fazendo proveito dos recursos da própria máquina, do próprio sistema operacional. A biblioteca do *Windows* é vasta e extensa, portanto recomendo a leitura de literaturas da área de informática no que tange Sistema Operacional. Depois livros técnicos de programação, e treino, muito treino.

Forte abraço e até mais. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução à multicamadas

Aprenda os conceitos e como construir suas primeiras aplicações distribuídas

Resumo DevMan

Aplicações distribuídas são amplamente utilizadas em ambientes corporativos. Aqui você aprenderá a criar suas primeiras aplicações para este tipo de arquitetura.

Nesse artigo veremos

- Introdução a multicamadas;
- Criando aplicações servidoras COM+;
- Criando aplicações cliente COM+.

Qual a finalidade

- Permitir a construção de sistemas distribuídos.

Quais situações utilizam esses recursos?

- Redes de computadores onde um único computador se encarrega de processar requisições clientes, ou, simplesmente, quando uma máquina precisa executar uma instrução remota na outra.

Guinther Pauli
(guinther@devmedia.com.br)
é Bacharel em Sistemas de Informação, desenvolvedor Delphi há 13 anos, certificado pela Borland em Delphi: Web, 6, 7, 2005, 2006 e Kylix, e pela Microsoft: MCP, MCAD, MCSD. NET, MCTS e MCPD. É editor-chefe das revistas ClubeDelphi e .net Magazine. Já trabalhou em grandes projetos multicamadas, envolvendo mais de 500 componentes servidores COM+ e 3000 aplicações clientes.

Com certeza boa parte dos leitores que estão iniciando em programação já se deparou com o termo "multicamadas", "computação distribuída", ou ainda, "objetos distribuídos". Confesso que quando comecei meus estudos em cima da tecnologia, ainda no *Delphi 3*, senti bastante dificuldade. São essas dificuldades, pelas quais passei, que pretendo esclarecer neste artigo.

No *Delphi*, a programação distribuída começou com o *Delphi 3*, como já comentei, mas foi a partir do *Delphi 4* que ganhou forças. Na época, a tecnologia foi batizada de MIDAS (*Multitier Distributed Application Services*). Com a versão 3 do MIDAS, lançada no *Delphi 6*, a tecnologia passou a se chamar *DataSnap*. Hoje ambos os nomes são sinônimos.

Neste artigo, veremos um pouco de conceitos, e logo a seguir, vamos à prática para demonstrar como construir aplicações distribuídas no *Delphi*. Enganam-se aqueles que pensam que o *DataSnap* só pode ser utilizado para construção de aplicações de banco de dados distribuídos, tanto que nesta introdução, faremos uso de um exemplo simples, sem acesso a BD.

O que é programação distribuída?

Imagine uma rede de computadores, conectados por uma LAN, rede WI-FI ou mesmo Internet. Suponha que existem terminais de rede com menos capacidade de processamento, com pouca memória, processador obsoleto etc. (*Pentium* 100 MHZ 16 RAM). E nesta mesma rede, existe um computador mais robusto, capaz de processar cálculos e requisições em milésimos de segundo (*Intel Core 2 Duo* 1.66 2 GB RAM).

Podemos então solicitar a essa máquina que realize determinado processamento para nós, e nos devolva apenas o resultado esperado. Tudo é feito pelo servidor. A máquina cliente funciona como uma espécie de "terminal burro".

Normalmente, essas solicitações são feitas através de chamadas remotas de procedimentos (o famoso RPC - *Remote Procedure Call*). Ou seja, invocamos um procedimento, uma função, só que ao invés dela ser executada localmente, é processada no servidor.

Para essa comunicação ser feita com sucesso, há a necessidade de utilização de um protocolo. No *Delphi*, temos suporte a vários protocolos de comunicação distribuída, onde cito: DCOM, MTS, COM+, SOAP, Web *Connection*, CORBA, *Sockets* etc. Algumas dessas tecnologias não são mais utilizadas, como por exemplo, o Web *Connection* e CORBA, que deram lugar ao SOAP. O DCOM/MTS deram lugar ao COM+.

Cada máquina envolvida no processo também é chamada de "camada". Daí o termo três camadas. Em nossos exemplos, como não teremos banco de dados, teremos duas camadas.

Muitos confundem a divisão de camadas lógicas e físicas e gostaria de esclarecer essa dúvida aqui. Uma divisão física é quando há mais de um computador envolvido. Melhor ainda, mais de um processo (do *Windows*) envolvido. Nesse caso, há uma comunicação entre processos (*out-of-proc*), onde aplicativos cliente e servidor devem trocar informações. Veja que nesse caso temos dois aplicativos.

Quando você pega, por exemplo, uma aplicação *Delphi* que possui um *DataModule*, podemos dizer que há uma separação lógica de duas camadas, pois o acesso a dados fica separado da interface de usuário. Mas na hora de compilar, vira um EXE só. Outro exemplo comum é uma aplicação ASP.NET, onde são criadas classes separadas que tratam exclusivamente do acesso a dados e ficam separadas da interface. É uma aplicação em camadas, porém lógica, pois ao compilar, vira tudo um processo só. Com isso, já sabemos o que são camadas físicas e lógicas.

Quando falamos em *DataSnap* com banco de dados, podemos dizer que temos uma arquitetura de três camadas. Isso porque teremos: 1 - servidor de banco de dados; 2 - servidor de aplicação; 3 - aplicação cliente. A **Figura 1** ilustra essa arquitetura.

Mão na Massa

Como de costume em meus artigos, sempre colocamos a mão na massa, ou seja, vamos ver isso tudo na prática. Tenha em mãos qualquer versão do *Delphi* posterior a 4 e vamos lá! Aqui usarei o *Delphi 7*. Vamos demonstrar de um dos principais protocolos da atualidade: o COM+ (para usar o COM+, é necessário um *Windows 2000, 2003, XP* ou *Vista*). O COM+ é largamente utilizado em *intranets*, enquanto o SOAP é mais recomendado para internet por utilizar o HTTP 80. Além disso, o COM+ é muito mais rápido que os demais protocolos, incluindo *Sockets*, para ser usado em *Intranets*.

Vamos imaginar o seguinte cenário. Nós temos uma máquina A com recursos limitados (também conhecida de *thin-client*), e um servidor robusto B (chamado também de *servidor de aplicação*). Agora pense que a máquina A precisa calcular o fatorial de um número, mas não possui processamento suficiente para isso (este é um exemplo didático). Então vamos criar uma aplicação cliente em A que solicite esse cálculo à máquina B e retorne o resultado. Precisamos criar então duas aplicações, a servidora que ficará na máquina B e a cliente (A), que chamará um procedimento remoto em B.

Criando o servidor COM+

No *Delphi 7*, clique em *File>New>Other* e localize na aba *ActiveX* o ícone *Transactional Object*. Isso cria um objeto do

Nota do DevMan

No .NET, o protocolo responsável pela comunicação é o .NET Remoting, nas versões 1.1 e 2.0 do framework. Na versão 3.0/3.5, um novo protocolo surgiu, como uma das bases do Windows Vista, que é o Windows Communication Foundation, ou simplesmente, WCF. O Delphi .NET dá suporte ao .NET Remoting, e em futuras versões certamente suportará o WCF.

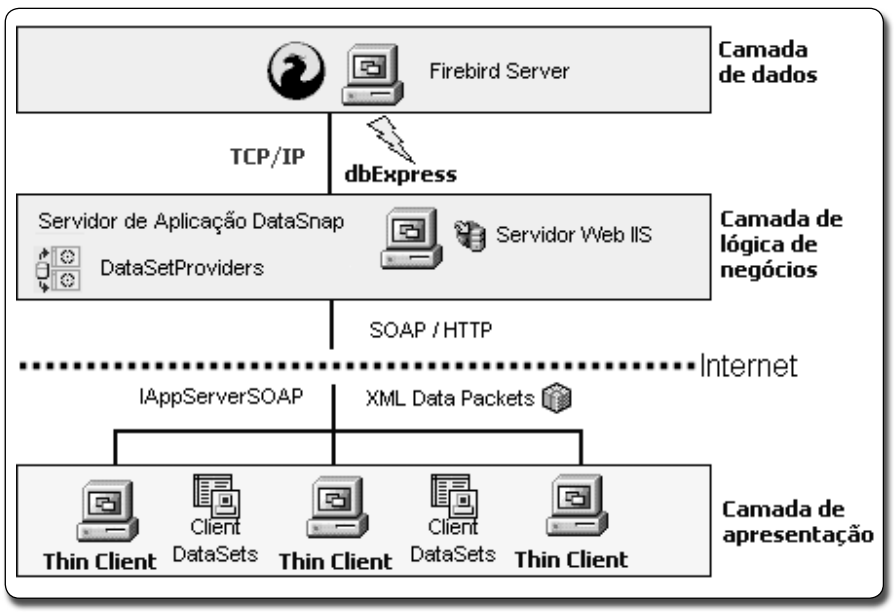


Figura 1. Exemplo de aplicação distribuída

tipo COM+ (**Figura 2**). Na próxima janela apenas digite *FatorialServer* (**Figura 3**) para a *coClass Name*.

Salve todos os arquivos do projeto em uma pasta chamada *Server*. A primeira tela que você vê agora se chama *Type Library*. Ela descreve as características do nosso objeto COM+. É aqui que definimos os métodos que o objeto suporta. Se reparar, há dois ícones nessa tela: *IFatorialServer* e *FatorialServer*.

O *IFatorialServer* é a interface do nosso objeto. Imagine uma interface COM+ como sendo um controle remoto de um *DVD Player*. Você pega o controle (micro A), senta no sofá, e comanda o DVD através desse controle. Mas o controle em si não faz nada, ele apenas repassa comandos para o *DVD Player* executar (B), através de um protocolo (nesse caso infra-vermelho ou mesmo *Bluetooth*). Em nossa aplicação, funciona da mesma forma, a máquina A usará a interface para acionar comandos no servidor. Quem realmente faz a coisa funcionar (o *DVD Player*), é o objeto representado pelo *FatorialServer*, também conhecido como *coClass*. Uma *coClass*, por definição, é uma classe que implementa uma interface.

Antes de continuar, vamos fazer uma pequena alteração. Feche o editor da *Type Libray* e acesse o menu *Tools>Enviroment Options>Type Library>Language* e troque para *Pascal* (**Figura 4**). Isso muda os tipos que irão aparecer na *Type Library*. Confirme com Ok.

Vamos continuar nosso exemplo. Clique em *View>Type Library*, dê um clique de direita em *IFatorialServer* e escolha a opção *New Method*. Dê ao novo método o nome de *Fatorial* (**Figura 5)**. Vá agora para a guia *Parametres*. Em *Return type* (tipo de retorno) escolha *Integer*. Agora logo abaixo digite um parâmetro chamado "x" do tipo *Integer*, como mostra a **Figura 6.**

O que acabamos de fazer aqui foi criar um método no micro B que pode ser invocado remotamente. Basta passar um valor inteiro como parâmetro que o método processará o cálculo no servidor e retornará apenas o valor para a aplicação cliente (A).

Salve todos os arquivos. Neste momento o *Delphi* gerou o código para o nosso objeto servidor (*coClass*), e também para a interface (dentro da *Type Library*). O que precisamos fazer agora é implementar o que faz o método *Fatorial*. Como uma interface não faz nada, apenas define, precisamos implementar esse cálculo na *coClass*, ou seja, no nosso objeto *FatorialServer*. Clique em *View>Units* e localize a única *unit* criada para o projeto, em nosso exemplo, como não dei nome, *unit1*.

Localize agora a implementação do método *fatorial*, abaixo da seção *implementation*, e implemente conforme a **Listagem 1,** que contém o código completo da *coClass*. Observe que nossa classe *TFatorialServer* herda de *TMtsAutoObject* (podemos dizer que o MTS é uma parte do COM+) e implementa nossa interface *IFatorialServer*. Voltando a nossa analogia, um *DVD Player* deve ser capaz de entender os comandos enviados por um controle remoto. Se ambos os lados (A e B), entenderem as regras do jogo (interface), então eles podem se comunicar.

Regisrando o servidor COM+

Para que o servidor *FatorialServer* possa ser acessado, ele precisa ser registrado no *Windows*, caso contrário, a aplicação cliente não saberá como encontrá-lo. Existem duas formas de registrar um objeto COM+: usando os *Serviços de Componente* nas *Ferramentas Administrativas* do *Painel de Controle* ou pelo próprio IDE do D7. Usaremos a segunda opção.

Clique em *Run>Install COM+ Objects*. A tela da **Figura 7** aparecerá. Marque *FatorialServer* e em seguida em *Install into new Application* digite "ClubeDelphi". Clique em Ok e Ok. Se quiser ver o objeto instalado (**Figura 8)** acesse o *menu* que mencionei no *Painel de Controle*. Usuários do *Windows Vista* devem executar o comando:

```
%windir%\system32\comexp.msc
```

Criando o cliente COM+

O cliente basicamente não faz nada (esse é um dos fundamentos da programação distribuída), ele simplesmente faz o servi-

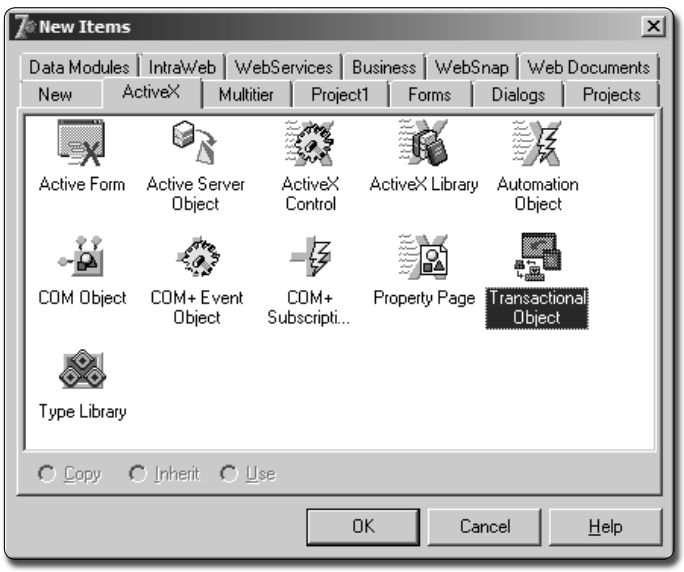


Figura 2. Criando um objeto COM+

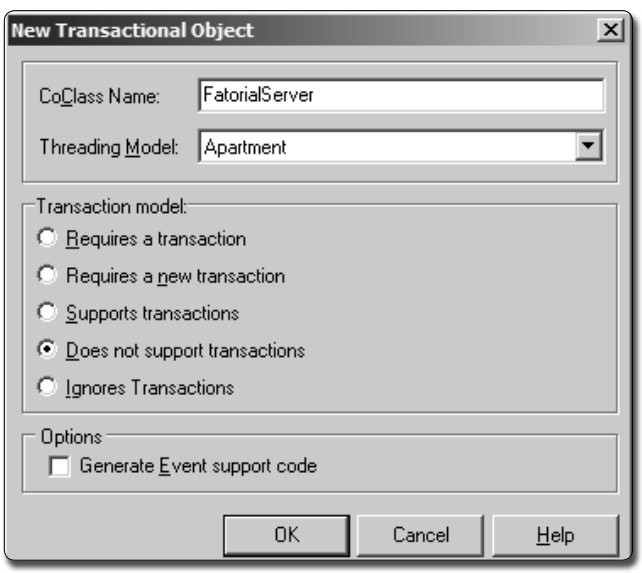


Figura 3. Fornecendo a coClass Name

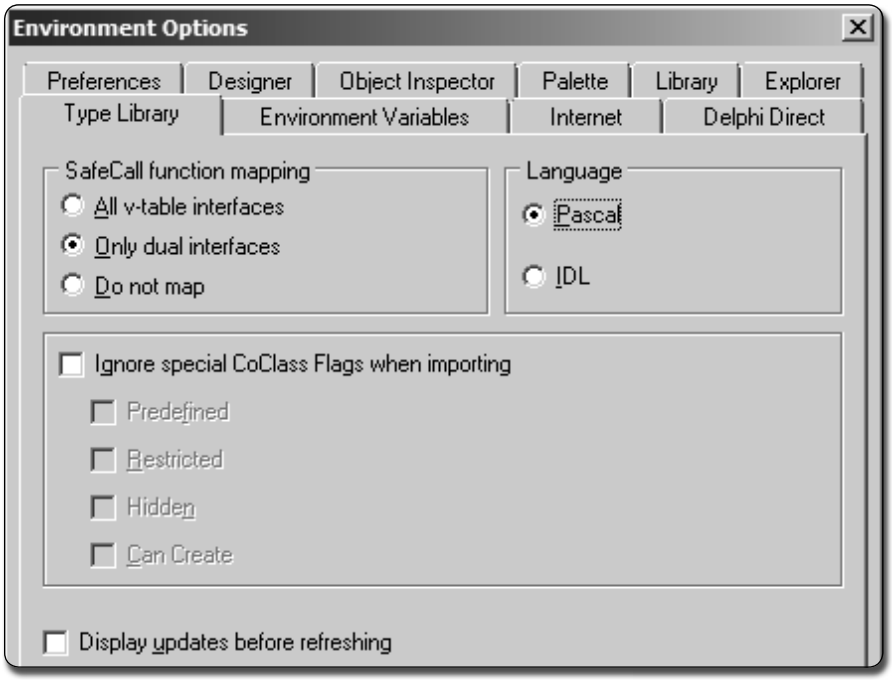


Figura 4. Definindo a linguagem da Type Library como Pascal

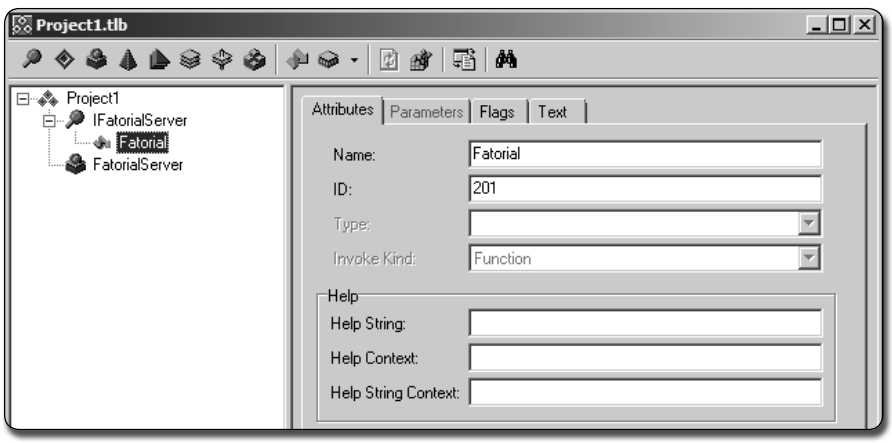


Figura 5. Criando um método na interface

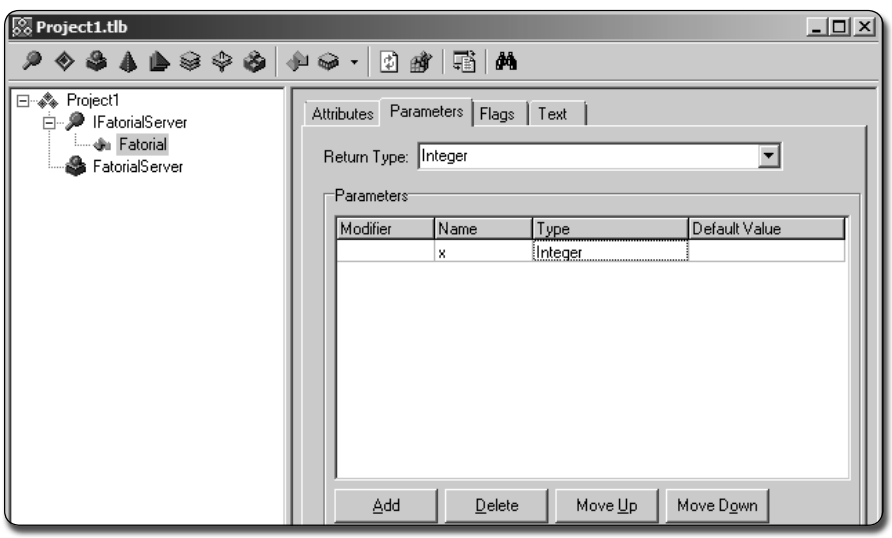


Figura 6. Definindo os parâmetros

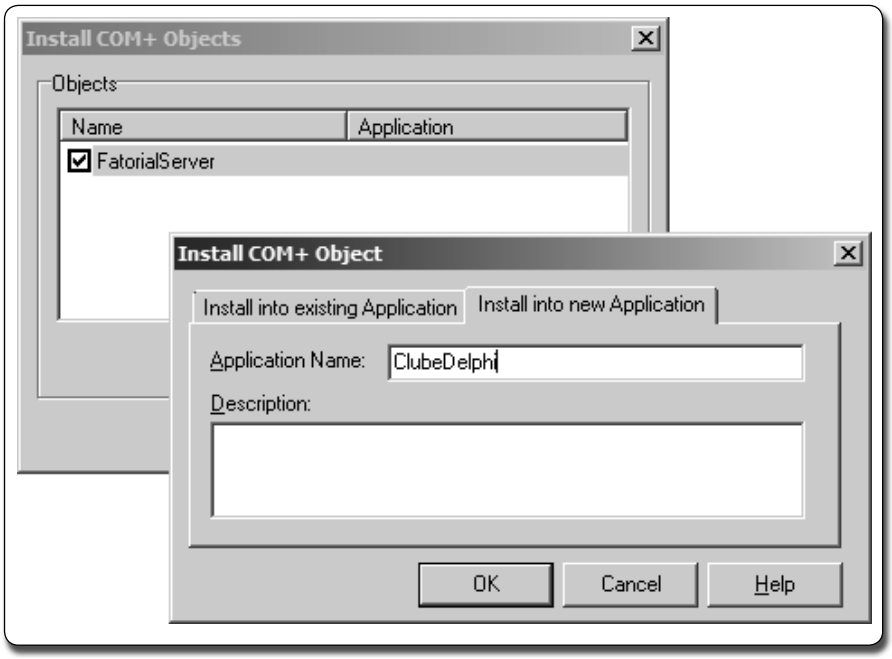


Figura 7. Instalando o objeto COM+

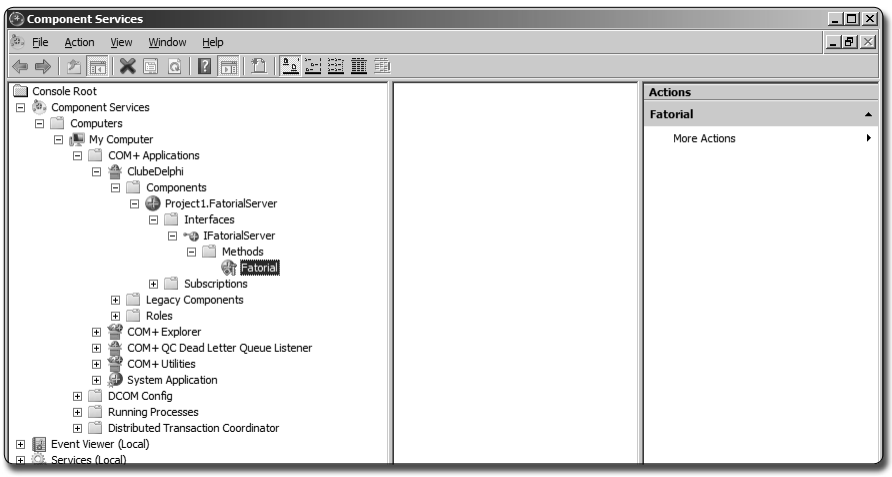


Figura 8. Componente instalado

dor trabalhar para ele. Em nosso cenário, a máquina A vai agora enviar uma chamada remota solicitando a B o cálculo do fatorial. Vamos ver como fazer isso.

Inicie uma nova aplicação no *Delphi* (a segunda camada), e salve todos os arquivos em uma pasta chamada *Client*. Clique em *Project>Import Type Library* para importar a interface que definimos no servidor. Aqui um aspecto importante: o cliente (máquina A) vai apenas enxergar a interface do servidor, e a interface não faz nada, apenas define. Logo, a lógica do negócio, a regra da fatorial, fica apenas de conhecimento do servidor (esse é outro fundamento da programação distribuída).

Na lista que aparece localize o nome do nosso servidor e deixe marcada a opção *Generate component Wrapper* (**Figura 9**),

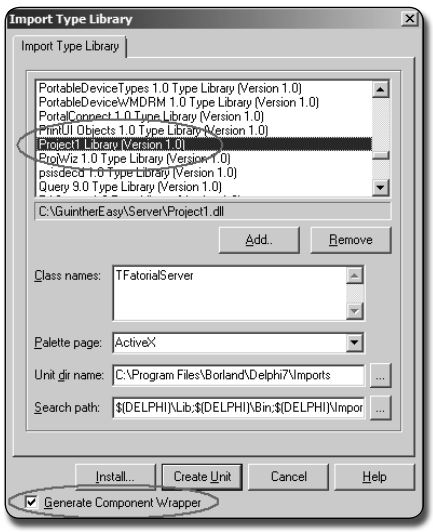


Figura 9. Importando a Type Library

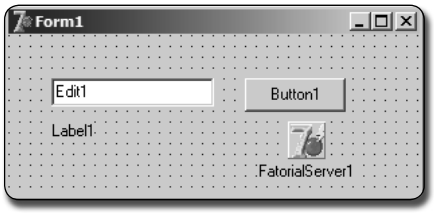


Figura 10. Tela principal do thin-client

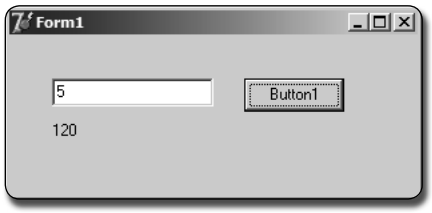


Figura 11. Aplicação em execução

Figura 12. Máquina cliente invocando máquina servidora

Listagem 1. CoClass com método Fatorial implementado

```
unit Unit1;
{$WARN SYMBOL_PLATFORM OFF}
interface
uses
  ActiveX, Mtsobj, Mtx, ComObj, Project1_TLB, StdVcl;
type
  TFatorialServer = class(TMtsAutoObject, IFatorialServer)
  protected
    function Fatorial(x: Integer): Integer; safecall;
  end;
implementation
uses ComServ;
function TFatorialServer.Fatorial(x: Integer): Integer;
var
  i,r: integer;
begin
  r := 1;
  for i := x downto 1 do
     r := r * i;
  result := r;
end;
initialization
  TAutoObjectFactory.Create(ComServer, TFatorialServer, Class_FatorialServer,
    ciMultiInstance, tmApartment);
end.
```

Listagem 2. Invocando o método remoto

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,r: integer;
begin
  x := StrToInt(Edit1.Text);
  r := FatorialServer1.Fatorial(x);
  Label1.Caption := IntToStr(r);
end;
```

isso vai facilitar nossa vida. Clique em *Install*, depois Ok e *Yes*.

Agora desenhe uma tela semelhante à **Figura 10**, onde temos os controles *Edit*, *Button* e *Label*, todos da paleta *Standard*. Agora vá até a paleta *ActiveX* e note algo interessante. Como marcamos a opção *Generate component Wrapper* ao importar a *Type Library*, o *Delphi* criou para nós um componente que encapsula o acesso ao servidor COM+. Tudo transparente. Coloque esse componente no formulário. Agora fica fácil, dê um duplo clique no botão e digite o código da **Listagem 2.** Execute e teste a aplicação (**Figura 11**).

> **Dica**: Se precisar recompilar a aplicação servidora e ela ainda estiver em memória, vá até o gerenciador do COM+ e dê um *Desligar (Shut Down)* na aplicação *ClubeDelphi*.

Executando remotamente

Até agora, utilizamos a mesma máquina para ambos cliente (A) e servidor (B), apesar de serem dois processos. Mas como fazer isso funcionar em uma *intranet* real, com duas máquinas físicas? Pois bem, aproveitei minha infra-estrutura de rede em meu escritório para realizar os testes. Ao executar a aplicação em outro terminal, obtivemos um erro, claro, pois o cliente não sabe onde está o servidor e nem como acessá-lo ("Classe não registrada"). Mas o procedimento é simples:

1 - No servidor, compartilhe o disco ou o diretório onde está a aplicação;

2 - Na máquina cliente, por compartilhamento de rede, mapeie o diretório/disco compartilhado no passo 1;

3 - Na máquina cliente, digite o seguinte no *prompt* para registrar a *Type Library*:

```
tregsvr z:\GuintherEasy\Server\Project1.dll
```

Onde z é o disco onde está aplicação e o restante o caminho da DLL servidora. *TRegSvr* é um aplicativo que vem com o *Delphi* e pode ser copiado para o local onde você vai executar o comando no *prompt*.

Pronto! Temos o cliente acessando o servidor, através de uma LAN, e a chamada remota sendo executada perfeitamente (**Figura 12**).

Conclusão

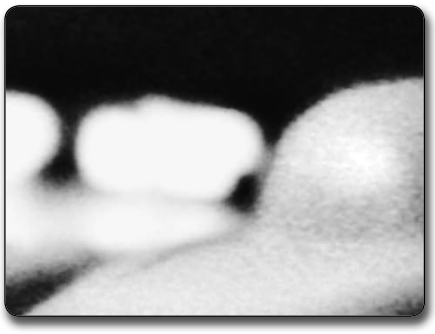
A programação distribuída com o *Delphi* envolve passos relativamente simples. Criamos exemplos com chamadas remotas com apenas alguns cliques e códigos. Caso tenha se interessado pelo assunto, não deixe de conferir meu artigo na edição 70, onde falo de *DataSnap* com banco de dados e SOAP para internet. Grande abraço a todos e sucesso com o *Delphi*. ●

Dê seu feedback sobre esta edição!

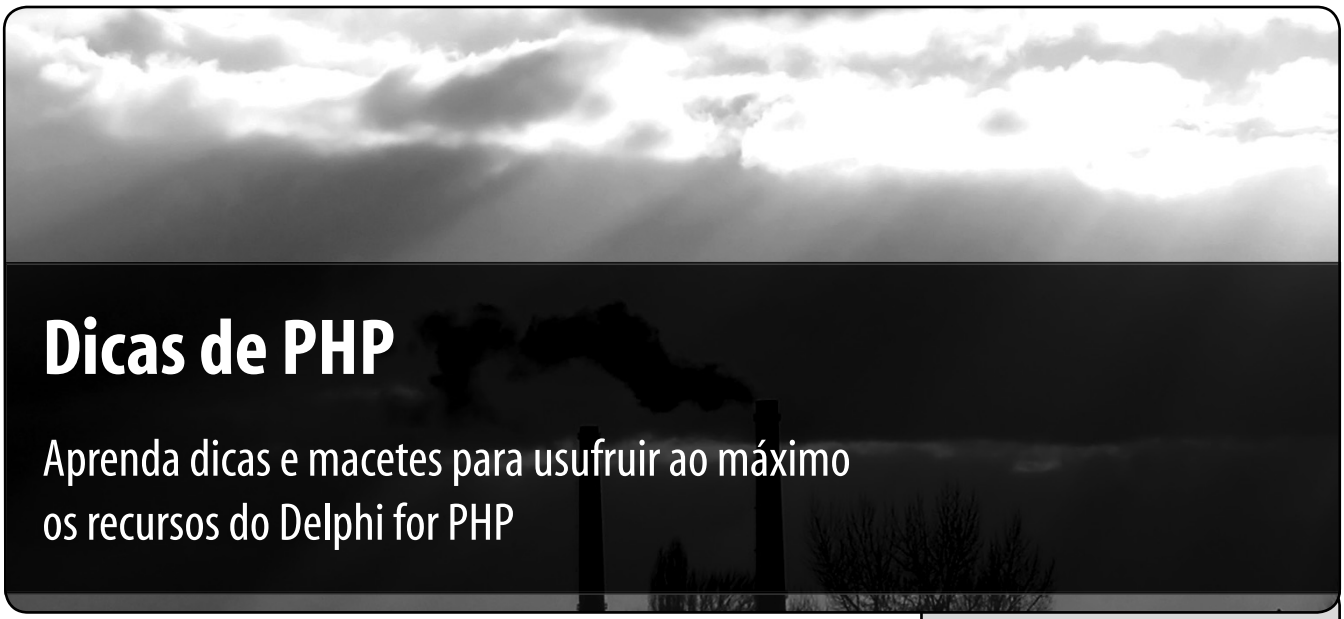
A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP



Dicas de PHP

Aprenda dicas e macetes para usufruir ao máximo os recursos do Delphi for PHP

Resumo DevMan

Veremos neste artigo alguns dos principais aspectos e características da linguagem PHP bem como do Delphi for PHP. Aprenderemos dicas valiosas para melhorar e agilizar ainda mais o desenvolvimento de aplicações PHP como o auxílio do IDE Delphi for PHP;

Nesse artigo veremos

• Dicas diversas sobre Delphi for PHP.

Qual a finalidade

• As finalidades são muitas e vão desde a criação do site em Delphi for PHP até sua publicação.

Quais situações utilizam esses recursos?

• Podemos utilizar as dicas aqui empregadas para uma série de situações.

Fabrício Desbessel (fabricio@fabricio.pro.br) é professor de Linguagem de Programação do Curso Técnico em Informática do Centro Tecnológico Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

Cada vez mais tenho a certeza que o *Delphi for PHP* veio para revolucionar o desenvolvimento Web. Com certeza a utilização do PHP é mais fácil e prática para a maioria das empresas que não possuem infra-estrutura suficiente para aplicações .NET. E também, é claro, muitas vezes os requisitos do negócio não compensam altos investimentos para fazer o básico.

Desde que comecei a utilizar o *Delphi for PHP* e escrever artigos, recebo diariamente inúmeros e-mails com dúvidas ou questões que, na maioria das vezes são simples de solucionar, mas infelizmente ainda temos poucos fóruns e documentação sobre o *Delphi for PHP*. Portanto, nesse artigo, escrevo algumas dicas que poderão facilitar a vida de desenvolvedores que estão começando a trabalhar com o *Delphi for PHP*.

1ª - Update 2

Antes de tudo é importante salientar que os *Update's* são essenciais para termos o melhor do produto em nossas mãos. No *Update 2* do *Delphi for PHP* tem-se uma serie de melhorias na IDE, correções de *Bug's* e, o principal, uma IDE mais ágil. Outro fator importante é a adição de novos componentes que foram sugestões de usuários.

Para fazer a instalação do *Update 2* é recomendável a *desinstalação* do *Delphi for PHP* existente na máquina. Pode-se baixar o arquivo de instalação no link

www.codegear.com/downloads/regusers/delphiphp. Depois de baixar é só instalar. Se você pretende experimentar o *Delphi for PHP* pode baixar o *Trial* que já está atualizado com o *Update* 2.

Depois de instalar o *Update* 2, é necessário adicionar manualmente os novos componentes. Isso é bem simples, bastando ir no menu *Component>Packages*, clicar no botão *Add* e selecionar o arquivo do pacote. Os novos componentes estão disponíveis nos seguintes locais (**Figura 1**):

```
\vcl\packages\mysql.package.php
\vcl\jquery\jquery.package.php
\vcl\jsval\formvalidator.package.php
\vcl\PEAR\packages\pear.package.php
```

> **Nota**: No momento do fechamento desta edição, a CG acabara de lançar o *Delphi for PHP 2.0*.

2ª - Error using a property not defined

Bem, ao abrir meus projetos tive uma triste surpresa: os mesmos não estavam mais funcionando. Apresentavam o erro: *"Error using a property, not defined as a member"*

Primeiramente achei que era um *Bug* da VCL e baixei uma nova versão no site do projeto VCL. Mas mesmo assim o erro persistiu em aparecer. Revisando meu projeto, notei que estava chamando em um código um componente sem levar em conta o nome em *case sensitive*. Essa é uma grande diferença encontrada no *Update* 2 que não permite mais essa facilidade que temos no *Delphi 7* e outros. Portanto, muito cuidado ao escrever os códigos no *Delphi for PHP*, pois letras maiúsculas e minúsculas são diferentes.

3ª - Atualização da VCL

Essa dica é simples: sempre que existir uma nova versão estável da *VCL for PHP*, atualize seu *Delphi*.

Em diversos fóruns vejo reclamações de *Bug's* ou problemas relacionados à VCL. Muitos culpam a *CodeGear* e acabam desestimulando outros desenvolvedores a trabalhar com o *Delphi for PHP*. Por isso é importante lembrar que a VCL for PHP é um projeto *Open Source* e, como todos outros, possuem vários problemas, mas que são solucionados rapidamente e, por isso precisamos ficar atentos às atualizações. Portanto esteja sempre atento e visite regularmente o site: *www.qadram.com/vcl4php*.

4ª Conexão com banco de dados Firebird

Podemos sim conectar ao *Firebird*. Inclusive podemos mostrar o banco no *Data Explorer* como se fosse um banco *Interbase*, e proporcionando todas as facilidades de arrastar tabelas para o formulário, bem como campos e outros. Ao tentar criar a conexão com um banco de dados *Firebird* encontra-se algumas dificuldades que podem ser contornadas. Para esclarecimento, utilizarei o *Firebird 2.0*.

Crie uma conexão como se fosse *Interbase*. No *Data Explorer* clique com o botão direito no item *Interbase* e escolha *Register DataBase*. Preencha os campos necessários e clique em *Ok*. Na **Figura 2** temos os parâmetros da conexão preenchidos.

Ao tentar abrir a conexão criada tem-se o erro: *"connection rejected by remote interface"*. Para solucionar o erro basta copiar o arquivo GDS32.DLL da pasta *C:\Windows\System32* que é do *Firebird* para a pasta do *Delphi for PHP*, em *C:\Arquivos de programas\CodeGear\Delphi for PHP\1.0*. Com isso será possível trabalhar no *Data Explorer* normalmente. Claro, é necessário ressaltar que com essa substituição de DLL's pode-se ter problemas em conexões com o *Interbase*. Portanto é recomendado que se faça um backup da DLL existente na pasta do *Delphi for PHP*.

Agora, basta arrastar uma tabela para o formulário e teremos todos os componentes necessários para a conexão mais um *DBGrid*. Ao compilar a aplicação pode-se ter o erro: *"unavailable database"*. Isso acontece pela falta de registro do serviço

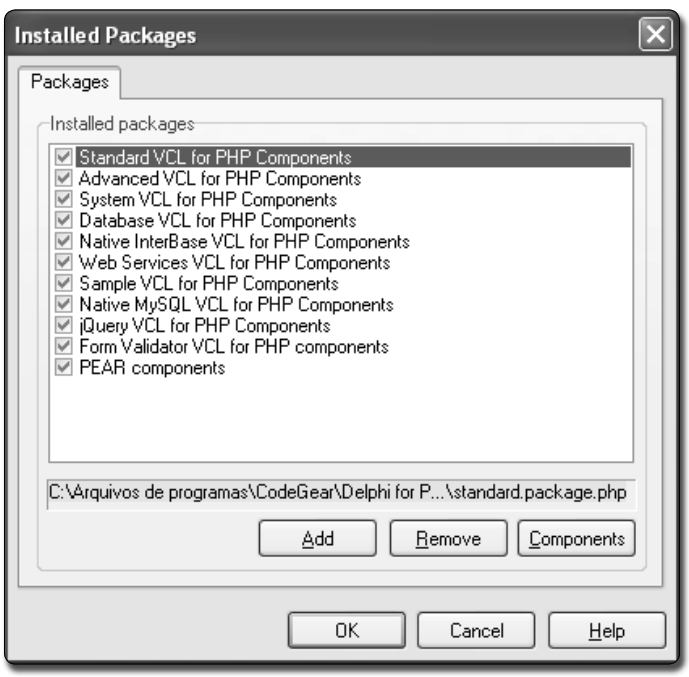


Figura 1. Instalação dos novos pacotes de componentes do Delphi for PHP

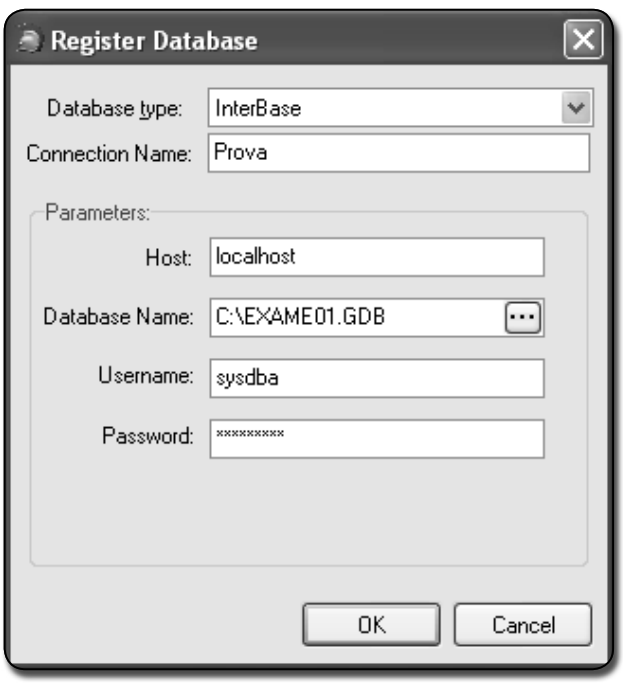


Figura 2. Conexão com o Firebird 2.0

no *Windows*, o que não é realizado pela instalação do *Firebird*. Então abra o arquivo *C:\WINDOWS\system32\drivers\etc\services* em um editor de texto e adicione a seguinte linha ao final do arquivo:

```
gds_db   3050/tcp    gds
# Firebird/Interbase
```

Além disso, pode-se ter outro erro ao compilar a aplicação: *"connection rejected by remote interface"*. É o mesmo erro encontrado ao tentar utilizar o *Data Explorer*, e acontece porque o servidor *Web Apache* também tem uma DLL de conexão (*GDS32.DLL*) que deve ser substituída. Portanto, copie o arquivo da pasta *System32* do *Windows* para a pasta *C:\Arquivos de programas\CodeGear\Delphi for PHP\1.0\apache2\bin*. Essa cópia deve ser realizada com o *Delphi for PHP* fechado.

Agora você pode compilar, que tudo deve funcionar como se fosse uma conexão com o *Interbase*. Lembrando que com essas substituições de arquivos, não é necessário alterar a propriedade *DiverName* do componente *Database*, deixando-a com *borland_ibase*.

5ª - Dicionário

Um recurso muito interessante do *Delphi for PHP*, que ainda é pouco utilizado é o Dicionário de Dados. O que esse recurso oferece é a possibilidade de deixar nossas tabelas e campos com nome mais sociáveis para os usuários finais.

Para criar um dicionário de dados acesse a propriedade *Dictionary* do componente *Database* e preencha com um nome que será dado à tabela que conterá o dicionário. Após preencher essa propriedade clique com o botão direito do mouse sobre o componente *Database* e escolha a opção *Create Dictionary*. Após a criação do dicionário, pode-se abrir a tabela e fazer a inclusão de registros com nomes mais fáceis de serem entendidos pelo usuário final. Ao abrir a tabela criada para armazenar o dicionário você terá os seguintes campos:

- *DICT_ID* = Informe um número seqüencial;
- *DICT_TABLENAME* = Informe o nome da tabela;
- *DICT_ FIELDNAME* = Informe o nome do campo;
- *DICT_PROPERTY* = Informe a propriedade *displaylabel*;
- *DICT_VALUE1* = Informe o nome mais fácil para o campo;
- *DICT_VALUE2* = Pode ser em branco.

Para exemplificar em um projeto de teste coloquei os valores dos campos da seguinte forma:

- *DICT_ID* = 1;
- *DICT_TABLENAME* = CACHORROS;
- *DICT_FIELDNAME* = CODCACHORRO;
- *DICT_PROPERTY* = *displaylabel*;
- *DICT_VALUE1* = Código do Cachorro.

Depois de criar e adicionar o dicionário de dados, todos *DBGrids* que estão ligados ao *Database* utilizarão os nomes dos campos conforme os valores *DICT_VALUE1*, e o mais importante é que seu dicionário de dados estará embutido na sua aplicação, facilitando também a vida do desenvolvedor.

6ª - Instruções SQL de forma direta

Às vezes é necessário executar alguma instrução diretamente no banco de dados, e que não faz sentido adicionar todos componentes de acesso como *Table* ou *Query*. Um exemplo disso, seria uma instrução para incrementar um contador ou uma instrução para fazer um *insert* em uma tabela etc.

O componente *Database* permite a execução de uma instrução *SQL* através da chamada do método *execute*. Para exemplificar crie um botão em sua aplicação e no seu evento *OnClick* coloque um código com uma instrução de *insert* como o seguinte:

```
$this->dbEXAME011->execute("Insert into
CACHORROS (CODCACHORRO,NOME,DATANASC,PE
SO,COR) Values (1,'Cachorro inserido',
'02-28-07', 5,'Branca')");
```

Claro que, no código anterior, você deve levar em conta o nome do seu componente *Database*, da sua tabela e de seus campos.

Essa possibilidade é muito interessante e possibilita que você automatize bastante sua aplicação, deixando mais tempo para seu usuário fazer realmente o que é necessário.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Fabrício Desbessel que mostra como trabalhar com Dicionário de Dados no Delphi for PHP.

www.devmedia.com.br/articles/viewcomp.asp?comp=5214&hl=

7ª - Criando um DBLookupComboBox

Infelizmente esse tão famoso componente, existente em várias versões do *Delphi*, não existe no *Delphi for PHP*. Mas podemos facilmente criar um, utilizando um simples *ComboBox* e algumas linhas de código.

Primeiramente vamos precisar de um componente *Query* com uma *SQL* para buscar os dados que serão listados como opção. Levando em conta que você está utilizando o banco de dados *Firebird* e o arquivo disponível no *download* desse artigo, coloque a seguinte instrução na propriedade SQL:

```
SELECT * FROM RACAS ORDER BY RACA
```

Adicione um componente *ComboBox* no formulário e no seu evento *OnBeforeShow* coloque o código da **Listagem 1**.

No código da **Listagem 1** tem-se uma interação até que a *Query* não chegue ao seu final, passando os valores, primeiramente para duas variáveis e depois adicionando como item no *ComboBox*.

É interessante não utilizar as propriedades *DataSource* e *DataField* do *ComboBox* porque ele tentará salvar no campo o valor apresentado e não seu identificador. No nosso exemplo, tentaria gravar a *RACA* e não o *CODRACA* como é o ideal. Por isso é necessário adicionar o valor no campo de forma manual. Para exemplificar como pegar o *ID* do *ComboBox*, coloque um *Button* e um *Label* na aplicação. Clique duas vezes sobre o botão e coloque o seguinte código:

```
$this->Label1->Caption=$this->
  ComboBox1->ItemIndex;
```

Listagem 1. Código para criar um DBLookupComboBox

```
while (!$this->Query1->EOF) {
  $id = $this->Query1->CODRACA;
  $raca = $this->Query1->RACA;
  $this->ComboBox1->AddItem($raca,null,$id);
  $this->Query1->Next();
}
```

A característica mais importante do *ComboBox* é que o seu *ItemIndex* é alterável e podemos utilizá-lo para colocar qualquer valor, seja número ou letras.

8ª - Utilizando Ajax

O *Delphi for PHP* traz embutido o *Framework Ajax* que facilita em muito nossa vida como programadores. Para usar a tecnologia *Ájax*, tem-se que utilizar os eventos da aba *Javascript* do *Object Inspector* e fazer chamada através do método *ajaxcall*.

Para fazer uma chamada precisamos antes ter uma função criada. O legal é que podemos, por exemplo, codificar o evento de um botão normalmente, testar e depois transformar sua execução em *Ajax*.

Para mostrar como funciona vou utilizar o botão inserido na 7ª dica. No seu evento *OnClick* da aba *Javascript* coloque o seguinte código:

```
echo $this->Button1->ajaxCall("Button2Click");
```

Nesse mesmo evento, abaixo do comentário *//Add your javascript code here* coloque o código:

```
return(false);
```

O *return(false)* impede que o *Postback* seja executado. Para finalizar, selecione o formulário e altere a propriedade *UseAjax* para *true*. Pronto, assim sua aplicação já estará utilizando a tecnologia mais eficaz na Web atual.

9ª DBPaginator é na verdade um DBNavigator

Recebo inúmeros e-mails diariamente questionando o funcionamento do *DBPaginator*. Todos acham que o mesmo serve para paginar uma quantidade de registros em um *DBGrid* ou *DBRepeater*, e não é isso que acontece.

A função real do *DBPaginator* no *Delphi for PHP* é a mesma que temos no *DBNavigator* dos *Delphi's*, que é a navegação entre registros.

Para exemplificar arraste alguns campos de uma tabela para o formulário. Lembre-se que se você quiser que o *Delphi for PHP* adicione *Edit's*, ao invés de *Label's*, que é o padrão, é necessário clicar no botão *Edit* existente na parte superior do *Data Explorer*.

Acrescente ao formulário um *DBPaginator* e ligue sua propriedade *DataSource* no mesmo *DataSource* dos *Edit's*. Altere as propriedade *CaptionFirst*, *CaptionLast*, *CaptionNext* e *CaptionPrevious* para "Primeiro", "Último", "Próximo" e "Anterior", respectivamente. Como isso você terá então um paginador de registros, conforme mostra a **Figura 3**.

10ª - Upload de arquivos

Com o *Update 2* tem-se um componente específico para essa finalidade. Seu nome é *Upload* e está localizado na paleta de componentes *Additional*.

Ao colocá-lo em um formulário tem-se, de forma automática, a definição do tipo de formulário HTML para *enctype="multipart/form-data"*, o que é necessário para poder fazer o envio de arquivos.

Para realmente enviar o arquivo, faz-se necessário a adição de um *Button* e no seu evento *OnClick*, a adição do código contido na **Listagem 2**.

No código da **Listagem 2**, primeiramente testa-se a existência de um arquivo selecionado para o envio, utilizando a função *$_FILES* existente no PHP e que permite buscar no nome do arquivo, sem seu *path*. Criou-se uma variável *$diretorio* para especificar o local destino do arquivo, onde optou-se por ser no diretório temporário da máquina, pois todos os usuários possuem permissão para gravar nesse diretório. Isso é muito importante cuidar na hora de publicar sua aplicação, pois, se o diretório destino não tiver permissão para gravação do usuário que está rodando o servidor, como por exemplo o *Apache*, seu *upload* não irá funcionar. Na variável *$nome_arquivo* colocou-se todo o seu *path* de destino acrescido do seu nome. E para fazer o *upload* utilizamos a função do PHP *move_uploaded_file*, passando o nome temporário do arquivo e seu destino completo. O nome temporário é um nome que o PHP dá ao arquivo temporariamente para recebê-lo e depois poder mover para o destino.

Se você não sabe o diretório onde fica sua aplicação, o que normalmente acontece quando hospedamos em um provedor, podemos utilizar a função *$_SERVER["DOCUMENT_ROOT"]* para descobrir o diretório que estamos, e com isso termos um diretório de arquivos dentro, alterando a linha *$diretorio* do código da **Listagem 2** para:

```
$diretorio = $_SERVER["DOCUMENT_ROOT"]."/arquivos/";
```

Além disso, ainda podemos verificar se o diretório existe e, se não existir criá-lo, com o seguinte código:

```
if (!is_dir($diretorio)){
  mkdir($diretorio);
}
```

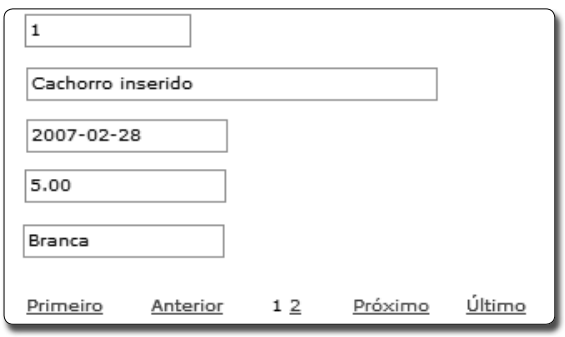


Figura 3. DBPaginator paginando os registros

Listagem 2. Código do botão Enviar arquivo (Upload)

```
if ($_FILES['Upload1']['name'] != "")
{
  $diretorio = "C:/Tmp/" ;
  $nome_arquivo = $diretorio . $_FILES['Upload1']
    ['name'];
  move_uploaded_file($_FILES['Upload1']['tmp_name'],
    $nome_arquivo);
}
```

Listagem 3. Código do evento OnBeforeShow do ListBox

```
$this->ListBox1->Clear();
$diretorio = $diretorio = "C:/Tmp/";
if (is_dir($diretorio)) {
  if ($dh = opendir($diretorio)) {
     while (($file = readdir($dh)) !== false) {
       if (filetype($diretorio.$file) == "file") {
          $this->ListBox1->addItem($file);
       }
     }
  }
}
```

11ª Listar arquivos de um diretório

Outra dica interessante é a iteração para listar arquivos em um diretório que poderá ser utilizada, por exemplo, para listar todas as fotos de um diretório, fazendo um *hyperlink* para as mesmas.

Coloque um *ListBox* no formulário, que será "populado" com o nome dos arquivos e, no seu evento *OnBeforeShow* coloque o código da **Listagem 3**.

Na **Listagem 3** definiu-se uma variável diretório que servirá de local para a busca de arquivos. A função *is_dir* verifica se o parâmetro passado é um diretório existente no servidor. A função *opendir* atribui a abertura de um diretório para uma variável que poderá sofrer iteração através da função *readdir*, a qual retornará o nome dos arquivos, seguindo a ordem definida no sistema operacional. E, para finalizar, utiliza-se a função *filetype* para certificar-se que trata-se de um arquivo e não um sub-diretório e/ou um link.

12ª Variáveis do Servidor

Com o passar do tempo você precisará saber algumas informações como caminho do arquivo *.php* no servidor, nome do arquivo, endereço IP do usuário, etc. Para tanto existe a variável *$_SERVER* que é um *array* com várias informações importantes. Para utilizá-lo basta informar entre colchetes o que se deseja saber. Abaixo algumas possibilidades de utilização das variáveis de servidor:

- *PHP_SELF*: Nome do arquivo e caminho relativo à base do site;
- *SERVER_NAME*: Nome do servidor que está rodando o *script PHP*;
- *DOCUMENT_ROOT:* Diretório raiz da execução do *script*;
- *REMOTE_ADDR:* Endereço IP do usuário do site;
- *REMOTE_HOST:* Nome do *host* do usuário do site.

Por exemplo, para saber o diretório atual da página utilize:

```
$_SERVER["DOCUMENT_ROOT"]
```

13ª Variáveis de Sessão

É muito fácil controlar variáveis em sessão com o *Delphi For PHP*. Isso porque ele já inicia uma sessão na execução de uma página. As variáveis são utilizadas através de *$_SESSION* e podem ser atribuídas e utilizadas em qualquer momento.

Por exemplo, pode-se depois de um *login* guardar o nome do usuário e seu e-mail com o seguinte código:

```
$_SESSION['Nome'] = 'Fabrício Desbessel';
$_SESSION['Email'] = 'fabricio@fabricio.pro.
br';
```

14ª Adicionando componentes

Já podemos encontrar fornecedores de componentes para o *Delphi For PHP*, como é o caso da *JomiTech*. Em seu site (*www.jomitech.com*) é possível baixar componentes *Free*, componentes em uma versão *Lite* e a versão *Full*. No site da *CodeGear* os usuários registrados podem fazer o *download* de uma versão *Full* promocional, conforme acordo da *CodeGear* e *JomiTech*. Acesse em: *cc.codegear.com/Item/25025*.

A instalação é simples, e para exemplificar fez-se o registro na página da *JomiTech* para baixar os componentes *Free*. Após o registro recebe-se um e-mail com o *link* para o *download*. Depois de baixar é só executar a instalação. Na **Figura 4** podemos ver os componentes instalados.

15ª Buscando valores no DBGrid

Para finalizar vamos a uma dica bem simples e que pode quebrar um bom galho. Buscar um valor em uma célula do *DBGrid*. Para testar coloque um *DBGrid* na sua aplicação, devidamente ligada a uma tabela. Adicione um botão e no seu código coloque o seguinte:

```
$valor=DBGrid1.getTableModel().getValue(0,
  DBGrid1.getFocusedRow());
```

Com o código anterior você estará pegando o valor da primeira coluna da linha selecionada. Para buscar o valor de outra coluna, mude o valor de 0 ("zero") para o número da coluna desejada.

Conclusão

Espero que essas dicas sirvam para você e que motivem mais pessoas a trabalhar com o *Delphi for PHP*. Continuem mandando e-mails, seja questionando ou sugerindo temas para artigos. Por mais que não responda diretamente seu e-mail, estou sempre atento e lendo os mesmos, para saber o que escrever. Muito obrigado pela colaboração de vocês e continuem usando o *Delphi for PHP*! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

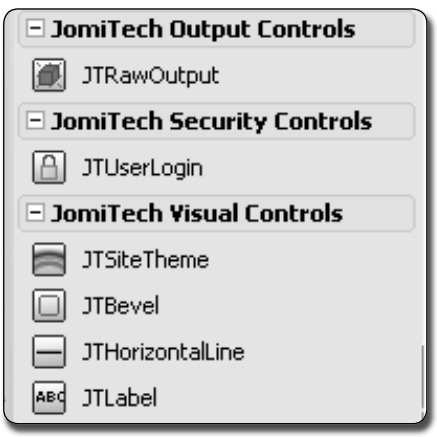


Figura 4. Componentes Free da JomiTech

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Entendendo os métodos GET e POST no PHP

Entenda e saiba como enviar e receber informações entre páginas usando os métodos GET e POST

Resumo DevMan

Neste artigo aprenderemos como enviar e receber dados de um formulário entre as várias páginas de um site. Além disso, com o exemplo criado, ainda poderemos ter um noção de como validar controles em tela utilizando esses métodos.

Nesse artigo veremos

- O que são métodos GET e POST;
- Qual a diferença entre ambos;
- Para que servem e como funcionam.

Qual a finalidade

- Entender como funciona o mecanismo de envio e recebimento de dados entre páginas através dos métodos GET e POST.

Quais situações utilizam esses recursos?

- A principal finalidade deste artigo é saber como enviar e receber dados de formulários de uma página para outra. Esse recurso é muito útil em telas de consultas ou cadastros.

No desenvolvimento Web, dois fatores importantíssimos contribuem bastante na hora de diferenciarmos do desenvolvimento *Win32*. Um deles diz respeito à segurança que é um item primordial na Web. Outro item bastante interessante faz menção à forma como as informações são transmitidas de uma tela (*página*) para outra. O formato da internet diverge bastante do formato *TForm-Based* (*Win32*), onde trabalhamos especificamente em memória.

Esses dois fatores serão discutidos neste artigo de forma bastante sucinta, mas rica em conceitos para que tenhamos total clareza dos fatos.

Desenvolveremos uma pequena aplicação onde usaremos os métodos *GET* e *POST* para enviar e receber, respectivamente, dados de um formulário para outra página.

Adriano Santos
falecom@adrianosantos.pro.br
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Mantém o blog Delphi to Delphi (www.delphi-todelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Entendendo todo o conceito

Um dos maiores paradigmas do desenvolvedor *Delphi* quando se depara com uma programação para Web, certamente é livrar-se dos "vícios" da programação *Win32*. Um desses vícios tem a ver com a forma como as coisas são tratadas na Web. Por exemplo: em *Win32* estamos

acostumados a declarar uma variável global e usá-la durante todo o ciclo de vida da aplicação, ou seja, ao finalizar o aplicativo todas as variáveis são automaticamente liberadas e destruídas. Esse paradigma enfrentado pelo programador *Delphi* torna o desenvolvimento Web um tanto lento devido às incapacidades do *HTML* e *HTTP*, por isso muitos programadores se aventuram e acabam esbarrando em problemas que são facilmente contornados seguindo as regras básicas da internet.

Já me deparei com vários desenvolvedores que querem fazer na Web exatamente como fazem em *Win32*, o que é um tremendo engano. Como diz meu amigo Guinther Pauli – *Um browser não é um formulário Delphi, infelizmente, então se acostume às limitações do HTML e HTTP*. E isso é um fato. Vejamos por que:

1. Cada página aberta, ou atualizada (*F5*), é processada como se fosse a primeira vez, o que chamamos de *stateless*;

2. Uma aplicação Web normalmente é pública e conseqüentemente vulnerável a ataques e *SQL Injection*;

3. Uma página, em tese, não enxerga os dados da outra. A menos que façamos uso de determinados mecanismos.

Com base nos itens vistos anteriormente, mais especificamente o último, veremos neste artigo como trabalhar com múltiplas páginas em *PHP*. Além disso, aprenderemos a diferença entre os métodos *GET* e *POST*.

Criando a aplicação

Vamos iniciar o desenvolvimento de dois arquivos para compor nosso *Web Site*. O Primeiro deles consiste em um cabeçalho com logotipo. Basta criar uma imagem semelhante à **Figura 1** e salvar como *topo.jpg* no mesmo diretório da aplicação. Em seguida abra o *Bloco de Notas* do *Windows* ou qualquer outro editor de sua preferência e digite o código da **Listagem 1**. Assim como fizemos em outros artigos sobre *PHP*, vamos usar um pouco de *HTML* para criar nossas páginas.

Salve o arquivo como *topo.htm*. O único segredo nesse código é a inclusão do arquivo *topo.jpg* através da *tag <img src>* do próprio *HTML*. Uma novidade em relação a outros artigos sobre *PHP* que vimos, é que a página de topo não é um *.php* e sim um *.htm*, ou seja, podemos usar também arquivos *HTML* comuns para o cabeçalho ao invés do *.php*. Nem todas as páginas de um *Web Site* dinâmico desenvolvido nessa linguagem precisam, necessariamente, ser salvas como *.php*.

Agora crie um novo arquivo no *Bloco de Notas* e vamos criar um rodapé simples. Nesse incluiremos uma tabela de três colunas e uma linha. Na primeira célula colocaremos um logotipo e nas demais os textos *DevMedia Group – Todos os direitos reservados* e *Copyright 2008*, respectivamente. Além disso, incluímos uma régua (*<hr>*) inferior para separar o *layout* de cabeçalho do corpo da página. Faça isso usando o código da **Listagem 2** e salve o arquivo como *bottom.htm*.

Não há muito que comentar nesse código. Estamos criando uma tabela usando a *tag <table>* e dentro dela uma linha (*<tr>*) com três células (*<td>*). Para ficar mais atraente o visual, criamos um estilo *CSS* (*<style>*) e codificamos o tipo de fonte e seu tamanho. Incluímos também uma barra superior para separar o rodapé do corpo da página.

Vamos preparar agora o *layout* da página principal que se chamará *cadastro.php*. Crie um novo arquivo e digite o código da **Listagem 3**.

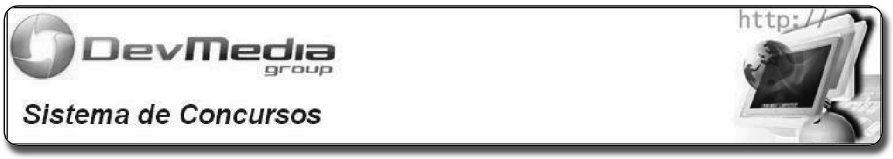


Figura 1. Topo do Web Site

Listagem 1. Código do arquivo topo.htm

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Topo</title>
</head>
<body marginheight="0" marginwidth="0" leftmargin="0" rightmargin="0">
  <img src="topo.jpg" width="750" height="120"><br>
  <hr align="left" width="750" size="1">
  <br>
</body>
</html>
```

Listagem 2. Código do rodapé do Web Site

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
<title>Bottom</title>
<style type="text/css">
<!--
.estilo{
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-size: 10px;
}
-->
</style>
</head>
<body>
  <hr align="left" width="750" size="1">
  <table width="750" border="0">
    <tr>
      <td width="190">
        <img src="devmedia.jpg" width="117"
          height="25"></td>
      <td width="343">
        <div align="center" class="estilo">
          DevMedia Group - Todos os direitos reservados
        </div>
      </td>
      <td width="203">
        <div align="right" class="estilo">
          Copyright 2008
        </div>
      </td>
    </tr>
  </table>
</body>
</html>
```

Listagem 3. Código da página cadastro

```
<?php
  include("topo.htm");
?>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
<title>Cadastro</title>
</head>
<body>
</body>
</html>
<?
  include("bottom.htm");
?>
```

Figura 2. Layout completo da página

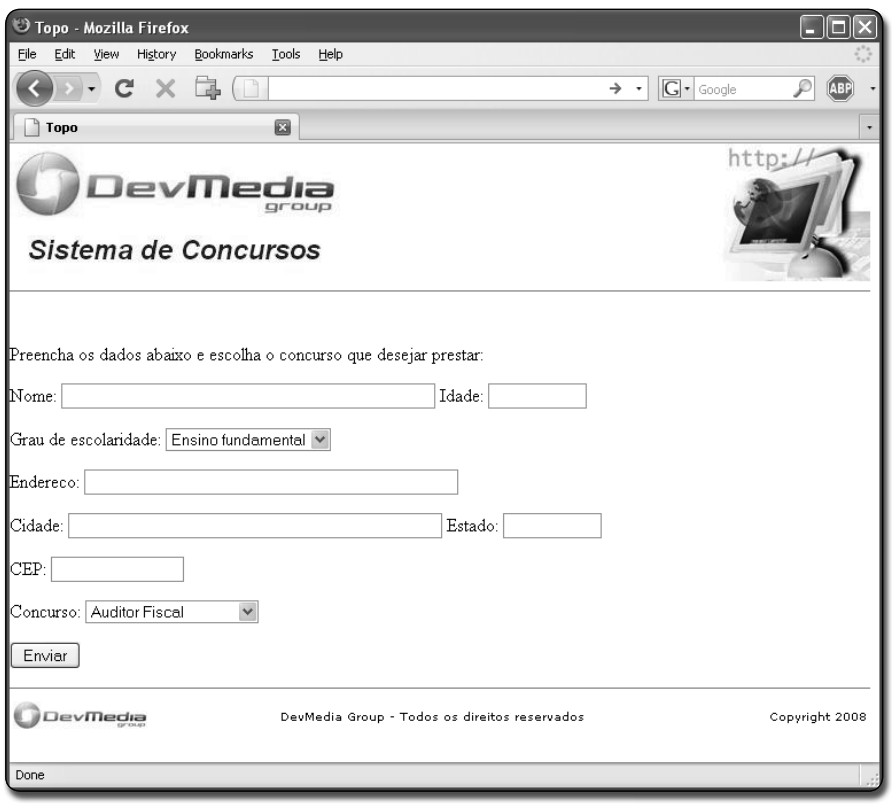


Figura 3. Layout do formulário para envio

Observe que já temos código *PHP* adicionado nessa página. No início e no fim, fizemos uma chamada ao método *include* do *PHP* que tem como finalidade, incluir o arquivo solicitado. Nesses dois casos *topo.htm* e *bottom.htm*. Salve como *cadastro.php*. O visual final do cabeçalho, rodapé e cadastro juntos se parecerá com a **Figura 2**.

Criando o formulário de envio

Muito bem, trabalhamos até agora somente a parte visual da página. Precisamos agora preparar o formulário de cadastro que enviará os dados para a página seguinte, onde o internauta fará a conferência dos dados. Sugiro que crie um layout como mostrado na **Figura 3**.

Vejamos como criar um *layout* como esse. Volte ao código da página *cadastro.php* e posicione-se entre as *tags* <*body*> e </*body*>. O *body* da página nada mais é que o corpo dela. Vamos criar um formulário usando a *tag HTML* <*form*>. A *tag* <*form*> nos obriga a declarar o atributo *method* e é ele o principal personagem do nosso artigo. Veja o código a seguir:

```
<form action="confirmacao.php" method="POST"
      name="frmPrincipal" id="frmPrincipal">
</form>
```

Em nosso exemplo enviaremos os dados da página atual (*cadastro.php*) para a página *confirmacao.php*. Em *HTML* temos duas formas de envio dos dados chamadas de métodos *GET* e *POST*. A diferença básica entre ambos é que o método *GET* envia os dados do formulário para a página seguinte escrevendo uma *string* no *URI* (*Uniform Resource Indicator*), mais conhecido como *URL*. O grande problema disso tudo é que o campo usado pelo *browser* suporta apenas *255* caracteres, além de ser inseguro. Já o método *POST* envia os dados para a página seguinte em forma de variáveis, o que se torna mais seguro e elegante. Para testar os dois métodos, vamos criar o formulário mostrado na **Figura 3**.

Como dito anteriormente, volte ao código da página *cadastro.htm* e encontre as *tags* <*body*> e </*body*>. Entre elas digite o código da **Listagem 3**.

Não entrarei em detalhes sobre o código *HTML* por não fazer parte do escopo deste artigo, porém uma informação valiosa nesse código precisa de uma atenção especial.

Como dito anteriormente, estamos usando a *tag <form>* para criar um formulário e enviar os dados para a tela seguinte. Para quem nunca usou formulário em *HTML*, a explicação é simples. A *tag <form>* precisa de alguns atributos para ser completa e eficaz. O primeiro deles é um dos mais importantes, o *action*.

Nele definimos para onde vão os dados coletados no formulário. Nesse caso estamos enviando para a página *confirmacao.php*. Em seguida configuramos o seu *method*, que é a forma como os dados serão enviados para processamento. Vamos testar primeiro com o método *POST* e depois com o *GET* para entendermos bem como funcionam.

Criando o arquivo de recepção dos dados

O principal objetivo do arquivo de confirmação é pegar os dados recebidos e imprimir em tela. Claro que podíamos fazer qualquer coisa com os dados, como validar as informações, gravá-las em uma tabela do banco de dados, gerar uma venda ou compra, enfim. Para fins didáticos, faremos apenas a sua impressão.

Para que possamos imprimir os dados em tela, precisamos descobrir como capturar tais informações. O *PHP* possui diversas variáveis globais que fazem a leitura constantemente de dados postados em formulários ou enviados por *URL*. Uma dessas variáveis, que acaba funcionando como se fosse um método, chama-se *$HTTP_POST_VARS*. Bem sugestivo para o que precisamos fazer. Vamos recapitular: na **Listagem 3** criamos um formulário para envio dos dados, e o método utilizado foi o *POST*, portanto a variável em questão encaixa-se como uma luva em nosso exemplo.

Para a impressão das informações poderemos utilizar um dos dois métodos: *echo* ou *print*. Nesse caso vamos usar *print*, mas ambas conseguem imprimir informações, variáveis, textos, elementos *HTML* etc. Veja o código da **Listagem 4**.

Note que a variável *$HTTP_POST_VARS* possui um parâmetro. Na verdade essa variável é um *array* que é alimentado sempre que uma requisição é processada. Cada elemento do *array* contém o valor de cada campo e pode ser impresso usando o próprio nome do campo, como podemos ver no código a seguir:

```
$HTTP_POST_VARS['txtNome'];
```

Se observar bem no arquivo *cadastro.php*, todos os elementos (*campos*) criados têm um nome atribuído no seu atributo *name*. Veja um exemplo:

```
<input name="txtNome" type="text"
id="txtNome" size="50" maxlength="50">
```

Estamos criando um *TextBox* (*type="text"*) com nome de *txtNome*. Esse nome é usado para capturar seu valor. No caso dos campos *slcGrau* e *slcConcuso* que são do tipo *ComboBox*, fazemos alguns *if's* para descobrirmos qual o item escolhido. Finalizadas as codificações, abra o *browser* e acesse o arquivo *cadastro.php*, complete os dados e confirme. Veja na **Figura 4** que os dados são carregados e impressos corretamente.

Nesse caso não há absolutamente nada de interessante. Os dados foram carregados com sucesso e impressos em tela.

Listagem 3. Código do corpo da página cadastro.php

```
<form action="confirmacao.php" method="POST" name="frmPrincipal" id="frmPrincipal">
  <p>Preencha os dados abaixo e escolha o concurso que desejar prestar:</p>
  <p>Nome:
    <input name="txtNome" type="text" id="txtNome" size="50" maxlength="50">
    Idade:
    <input name="txtNome" type="text" id="txtNome" size="10" maxlength="3">
  </p>
  <p>Grau de escolaridade:
    <select name="slcGrau" id="slcGrau">
      <option value="EF">Ensino fundamental</option>
      <option value="EM">Ensino Medio</option>
      <option value="ES">Ensino Superior</option>
    </select>
  </p>
  <p>Endereco:
     <input name="txtNome" type="text" id="txtNome" size="50" maxlength="50">
  </p>
  <p>Cidade:
     <input name="txtNome" type="text" id="txtNome" size="50" maxlength="50">
    Estado:
    <input name="txtNome" type="text" id="txtNome" size="10" maxlength="2">
  </p>
  <p>CEP:
     <input name="txtNome" type="text" id="txtNome" size="15" maxlength="9">
  </p>
  <p>Concurso:
     <select name="slcConcurso" id="slcConcurso">
       <option value="AF">Auditor Fiscal</option>
       <option value="DG">Delegado</option>
       <option value="FT">Fiscal de Transporte</option>
     </select>
  </p>
  <p>
    <input name="btnEnviar" type="submit" id="btnEnviar" value="Enviar">
  </p>
</form>
```

Listagem 4. Código de captura e impressão dos dados do formulário

```
Seus dados foram recebidos com sucesso. Verifique os dados abaixo:
<br>
<?
  print "<b>Nome:</b> " . $HTTP_POST_VARS['txtNome'] .
    " <b>Idade:</b> " . $HTTP_POST_VARS['txtIdade'] .
    "<br>";
  $grau = '';
  if ($HTTP_POST_VARS['slcGrau'] == "EF"){
    $grau = "Ensino Fundamental";
  }elseif ($HTTP_POST_VARS['slcGrau'] == "EM"){
    $grau = "Ensino Médio";
  }else{
    $grau = "Ensino Superior";
  }
  print "<b>Grau de escolaridade:</b> " . $grau . "<br>";
  print "<b>Endereço:</b> " . $HTTP_POST_VARS['txtEndereco'] . "<br>";
  print "<b>Cidade:</b> " . $HTTP_POST_VARS['txtCidade'] . " <b>Estado:</b> " .
    $HTTP_POST_VARS['txtEstado'] . "<br>";
  print "<b>CEP:</b> " . $HTTP_POST_VARS['txtCEP'] . "<br>";
  $concurso = '';
  if ($HTTP_POST_VARS['slcConcurso'] == "AF"){
    $concurso = "Auditor Fiscal";
  }elseif ($HTTP_POST_VARS['slcConcurso'] == "DG"){
    $concurso = "Delegado";
  }else{
    $concurso = "Fiscal de Transporte";
  }
  print "<b>Concurso:</b> " . $concurso . "<br>";
?>
```

Nota do DevMan

Além das variáveis $HTTP_GET_VARS e $HTTP_POST_VARS ainda podemos utilizar $_POST e $_GET para obter tais informações. Veja como poderíamos proceder no exemplo deste artigo modificando o código do arquivo confirmacao.php.

```
<?
  print "<b>Nome:</b> " . $_
POST['txtNome'] .
    " <b>Idade:</b> " . $_
POST['txtIdade'] . "<br>";

  $grau = '';
  if ($_POST['slcGrau'] == "EF"){
    $grau = "Ensino Fundamental";
  }elseif ($_POST['slcGrau'] == "EM"){
    $grau = "Ensino Médio";
  }else{
    $grau = "Ensino Superior";
  }
  print "<b>Grau de escolaridade:</b>
" .
    $grau . "<br>";
  print "<b>Endereço:</b> " . $_
POST['txtEndereco']
    . "<br>";
  print "<b>Cidade:</b> " . $_
POST['txtCidade']
    . " <b>Estado:</b> " . $_POST
['txtEstado'] . "<br>";
  print "<b>CEP:</b> " . $_
POST['txtCEP'] . "<br>";
  $concurso = '';
  if ($_POST['slcConcurso'] == "AF"){
    $concurso = "Auditor Fiscal";
  }elseif ($_POST['slcConcurso'] ==
"DG"){
    $concurso = "Delegado";
  }else{
    $concurso = "Fiscal de Transporte";
  }
  print "<b>Concurso:</b> " . $concurso
. "<br>";
?>
```

Vamos fazer um novo teste.

Retorne ao arquivo *cadastro.php* e localize a linha onde declaramos nosso *<form>*. Após encontrar, modifique o atributo *method* para *GET*. A linha deverá ficar como a seguir:

```
<form action="confirmacao.php" method="GET"
    name="frmPrincipal" id="frmPrincipal">
```

Apenas salve o arquivo e tente fazer o teste novamente. Naturalmente que os dados não serão capturados, já que necessitaremos alterar o código do arquivo *confirmacao.php*, mas um detalhe pode ser facilmente percebido. Olhe para a barra de endereços do navegador. Perceberá que uma grande *string* foi gravada nela. No meu caso ficou como segue:

```
http://www.adrianosantos.pro.br/confirmacao.
php?txtNome=Adriano+Santos&txtIdade=31&slc
Grau=ES&txtEndereco=Rua+HUM%2C+2000&txtCid
ade=S%E3o+Paulo&txtEstado=SP&txtCEP=08233-
000&slcConcurso=AF&btnEnviar=Enviar
```

Esse é o resultado do método *GET*. Todos os campos são passados como parâmetro na *URL*. O primeiro é passado por um caractere interrogação (?) e os demais pelo caractere *&*. Exemplo:

```
http://www.adrianosantos.pro.br/confirmacao.
php?Codigo=001&Nome=Adriano
```

Estamos passando o código e o nome do usuário. A variável global para pegar esse dado é *$HTTP_GET_VARS*, como já era esperado. Experimente retornar ao código do *confirmacao.php* e trocar todos os *$HTTP_POST_VARS* por *$HTTP_GET_VARS* e teste novamente. Com toda certeza o mesmo resultado será obtido.

Veja que alteramos bem pouco do código para adaptar nosso sistema ao uso de *GET* e *POST*.

Conclusão

De fato os métodos *GET* e *POST* possuem poucas diferenças. A principal diferença e que acaba implicando na questão de segurança, diz respeito à barra de endereços. Ela acaba recebendo não somente os nomes de campos, como também os valores digitados neles. Isso pode ser motivo e brecha de segurança. Tome cuidado ao usar métodos *GET* em seu *Web Site*.

Um forte abraço e até a próxima. ●

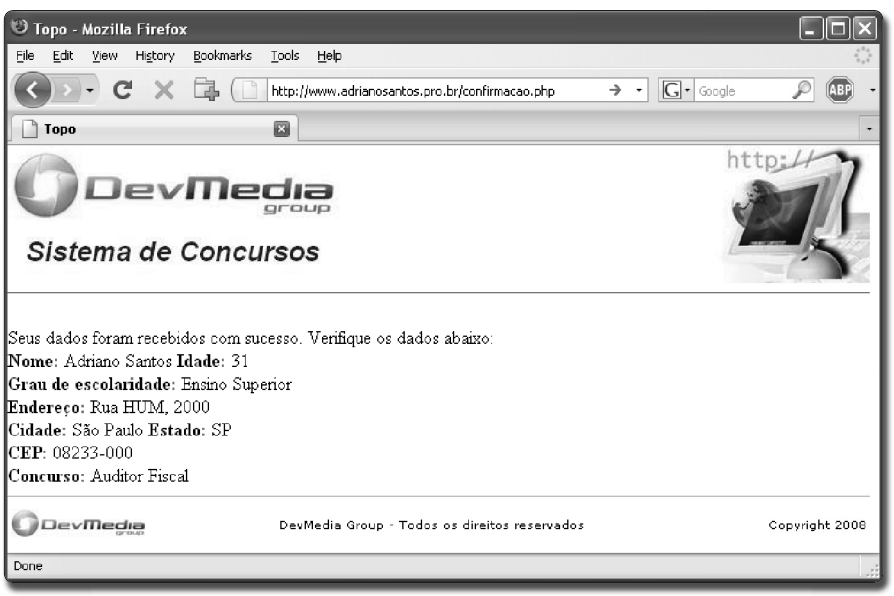


Figura 4. Cadastro confirmado

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

PLANO
PROFISSIONAL 1

- 3 domínios independentes
- 50 GB de transferência
- 30 GB de e-mails com SSL
- 1 GB de espaço
- Painel de controle
- ASP, ASP.NET 3.5 E PHP 5
- Access ilimitado
- 3 bancos MYSQL 5
- 1 banco SQL Server 2005 Express

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

Tudo isso por apenas R$ **25,90** por mês

Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS
1983–2008

SafeNet